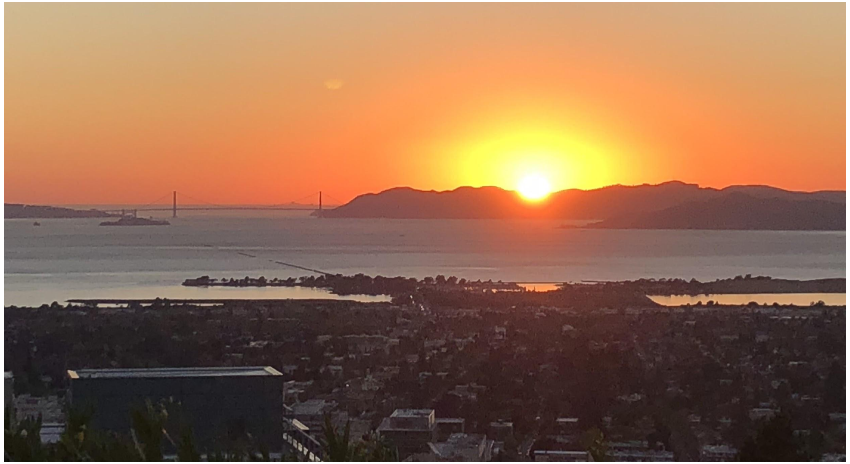
Neural Networks with Euclidean Symmetry for Physical Sciences

3D rotation- and translation-equivariant convolutional neural networks (for points, meshes, images, ...)



BERKELEY LAB

Tess Smidt
2018 Alvarez Fellow in Computing Sciences

CSA Summer Series
2020.07.01

Neural Networks with Euclidean Symmetry for Physical Sciences

3D rotation- and translation-equivariant convolutional neural networks (for points, meshes, images, ...)

Talk Takeaways

1. First a deep learning primer!
2. Different types of neural networks encode assumptions about specific data types.
3. Data types in the physical sciences are geometry and geometric tensors.
4. Neural networks with Euclidean symmetry can natural handle these data types.
   a. How they work
   b. What they can do

BERKELEY LAB

Tess Smidt

2018 Alvarez Fellow in Computing Sciences

CSA Summer Series
2020.07.02

A brief primer on deep learning

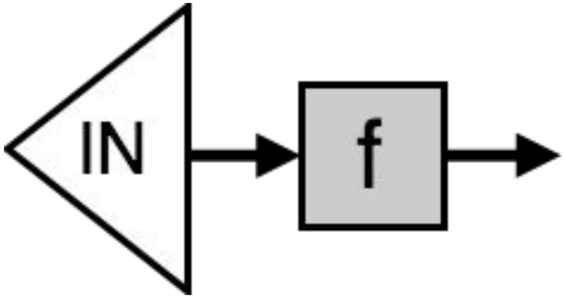
Skip?

deep learning ⊂ machine learning ⊂ artificial intelligence

A brief primer on deep learning

model ("neural network"):
Function with learnable parameters.

$$y = f(x)$$



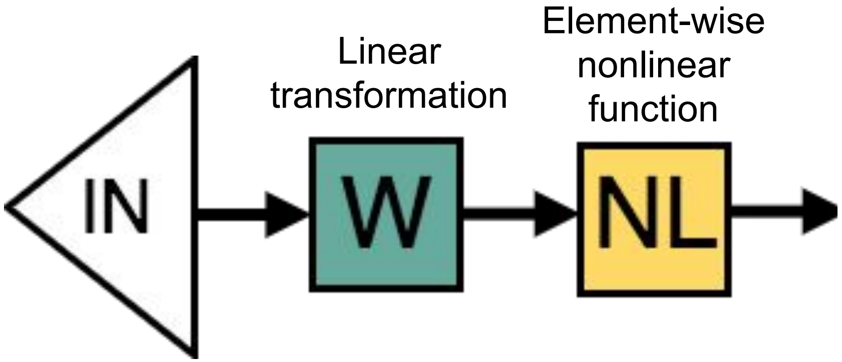
A brief primer on deep learning

model ("neural network"):
Function with learnable parameters.

Ex: "Fully-connected" network

$$y = \tanh(\boxed{W}x + \boxed{b})$$

Learned Parameters



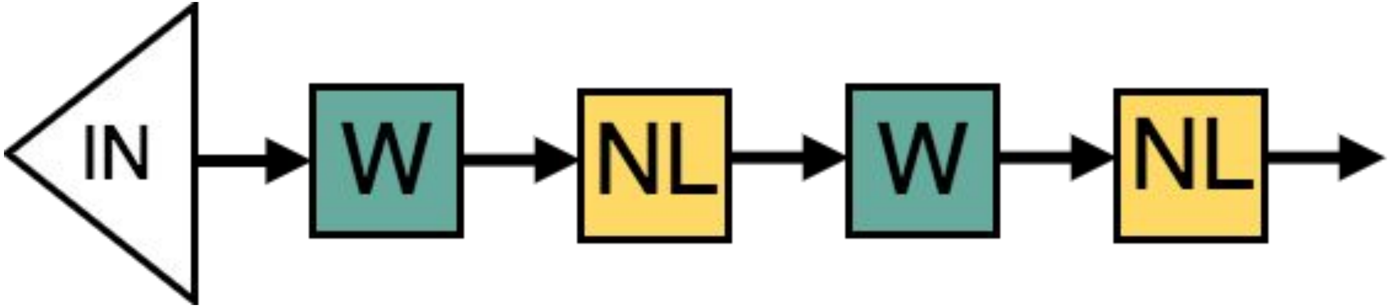
A brief primer on deep learning

model (“neural network”):
Function with learnable parameters.

Ex: "Fully-connected" network

$$y = \tanh(\boxed{W_2}\tanh(\boxed{W_1}x + \boxed{b_1}) + \boxed{b_2})$$

Learned Parameters



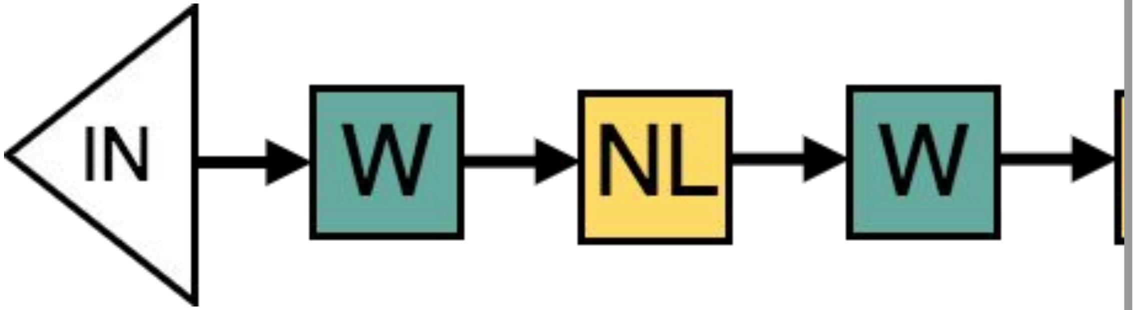
Neural networks with multiple layers can learn more complicated functions.

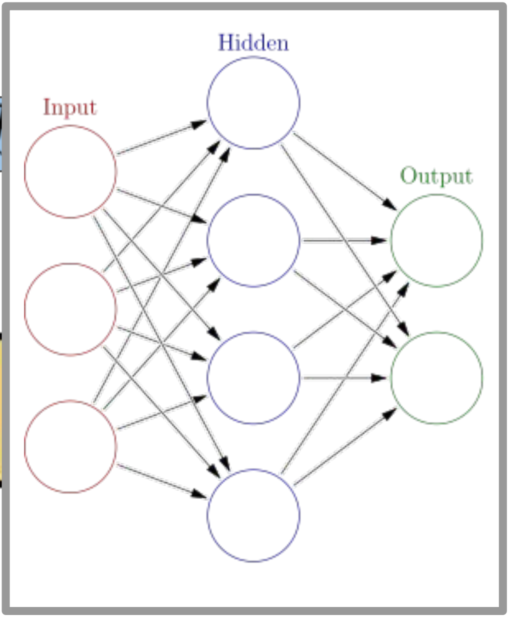
A brief primer on deep learning

model (“neural network”):
Function with learnable parameters.

Ex: "Fully-connected" network

$$y = \tanh(\boxed{W_2}\tanh(\boxed{W_1}x + \boxed{b_1}) + \ldots$$

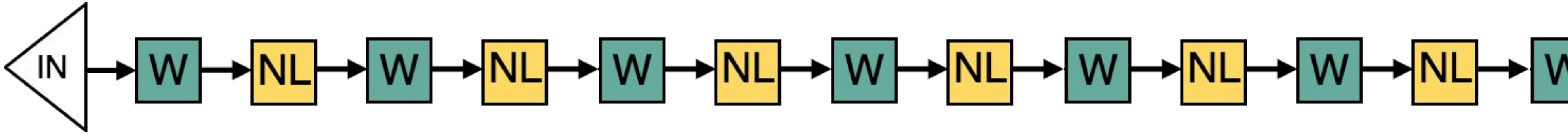




Neural networks with multiple layers can learn more complicated functions.

A brief primer on deep learning

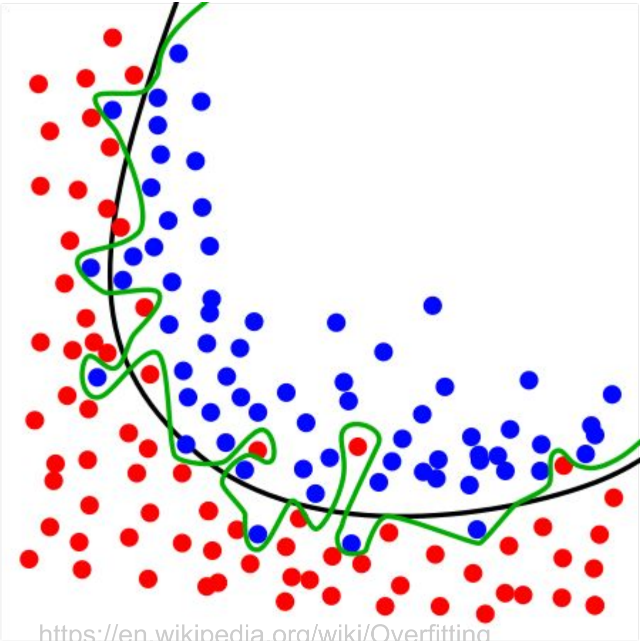
deep learning:
Add more layers.



A brief primer on deep learning

data:

Want lots of it. Model has many parameters. Don't want to easily overfit.



https://en.wikipedia.org/wiki/Overfitting

A brief primer on deep learning

cost function:

A metric to assess how well the model is performing.
The cost function is evaluated on the output of the model.
Also called the **loss** or **error**.

A brief primer on deep learning

way to update parameters:

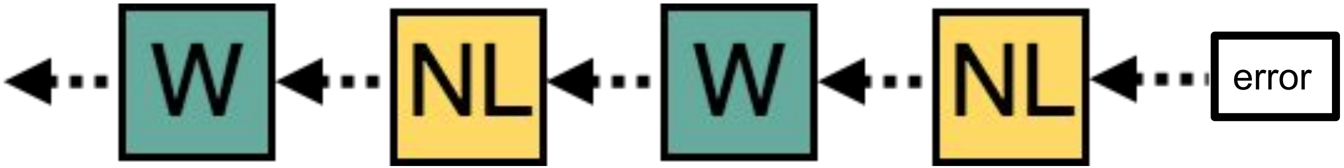
Construct a model that is differentiable

Easiest to do with differentiable programming frameworks: e.g. Torch, TensorFlow, JAX, ...

Take derivatives of the cost function (loss or error) wrt to learnable parameters.

This is called backpropogation (aka the chain rule).

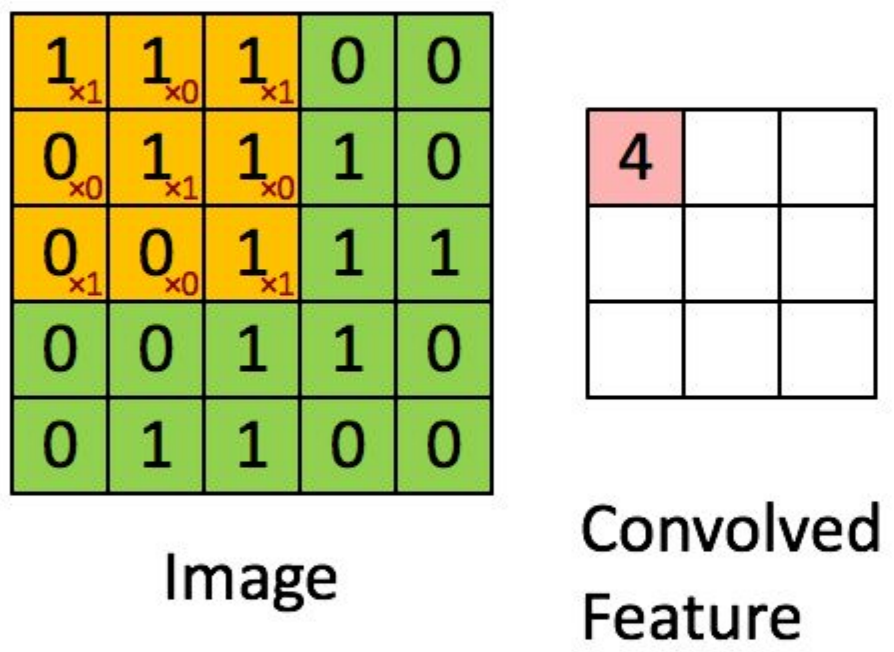
$$\Delta W_{ij} = -\eta \frac{\partial \; \mathrm{error}(f(W,x),y)}{\partial W_{ij}}$$



A brief primer on deep learning

convolutional neural networks:

Used for images. In each layer, scan over image with learned filters.



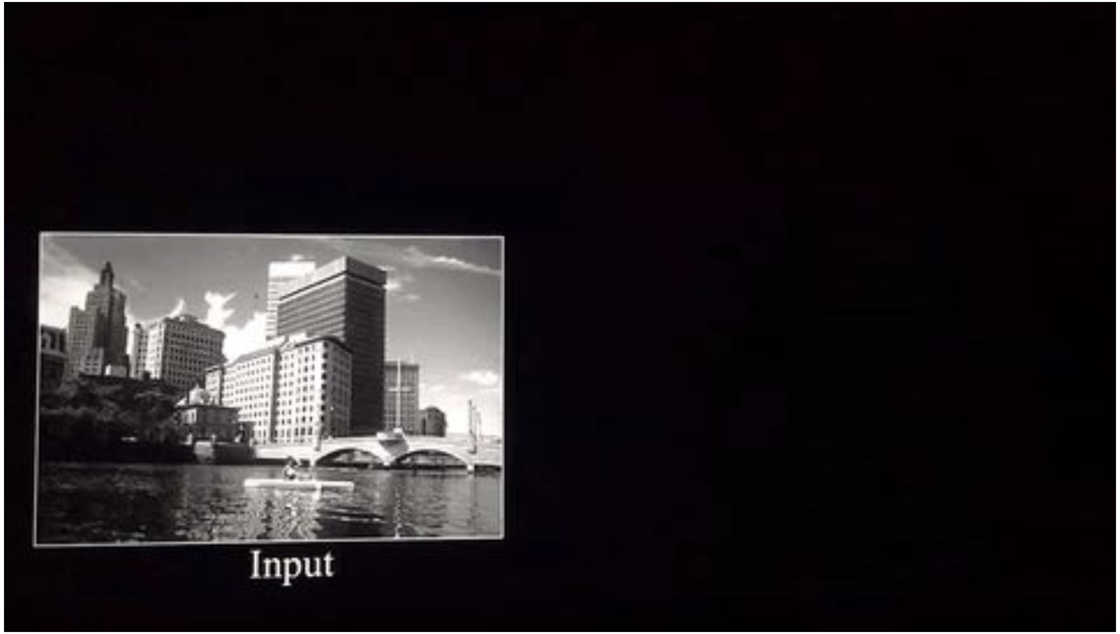
http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

A brief primer on deep learning

Back

convolutional neural networks:

Used for images. In each layer, scan over image with learned filters.

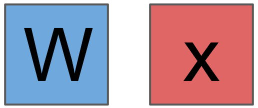


http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/

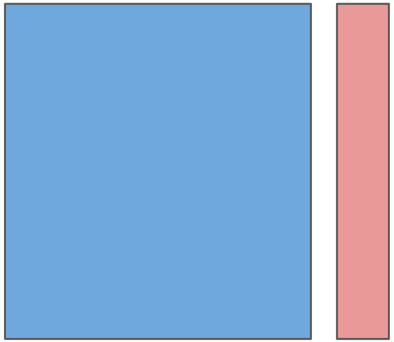
Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.

W x

Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.

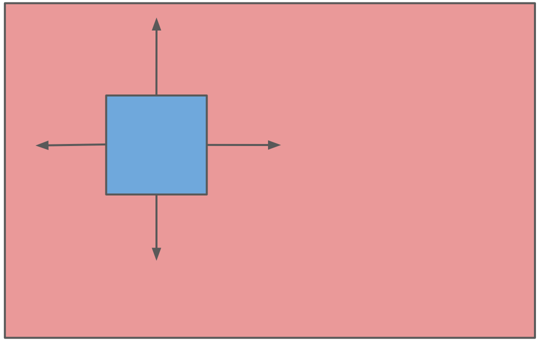


Arrays ➪ Dense NN



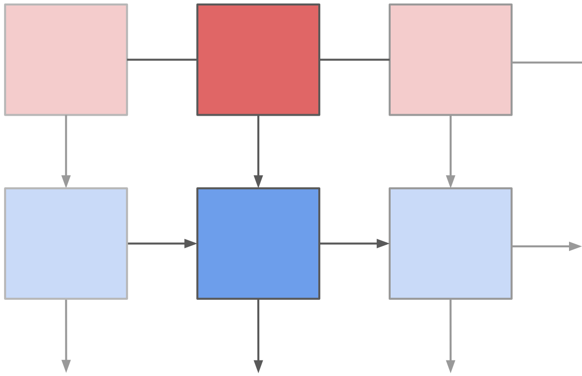
Components are independent.

2D images ➪ Convolutional NN



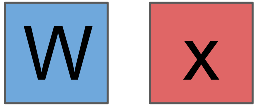
The same features can be found anywhere in an image. Locality.

Text ➪ Recurrent NN

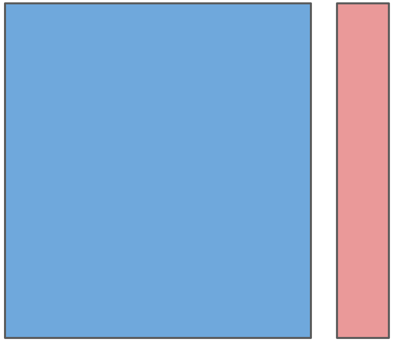


Sequential data. Next input/output depends on input/output that has come before.

Neural networks are specially designed for different data types. Assumptions about the data type are built into how the network operates.

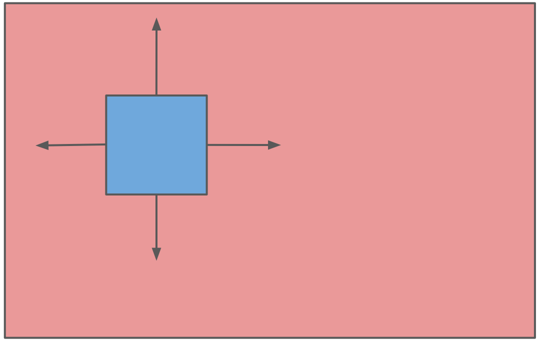


Arrays ➪ Dense NN



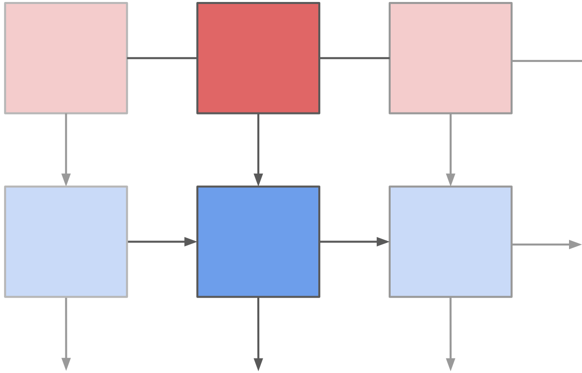
Components are independent.

2D images ➪ Convolutional NN



The same features can be found anywhere in an image. Locality.

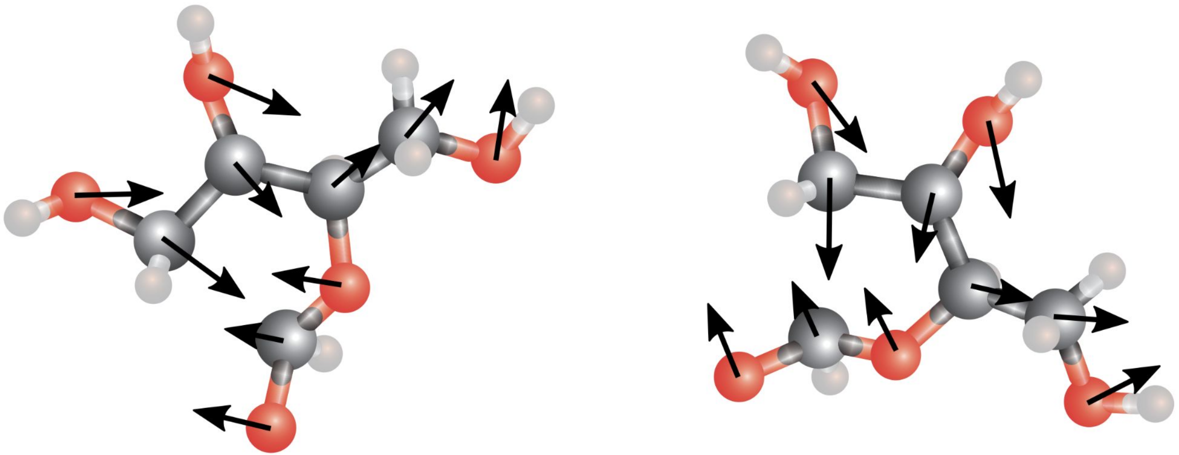
Text ➪ Recurrent NN



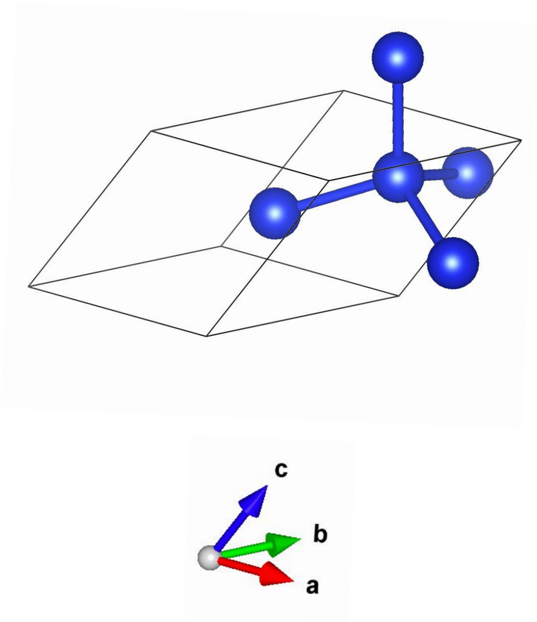
Sequential data. Next input/output depends on input/output that has come before.

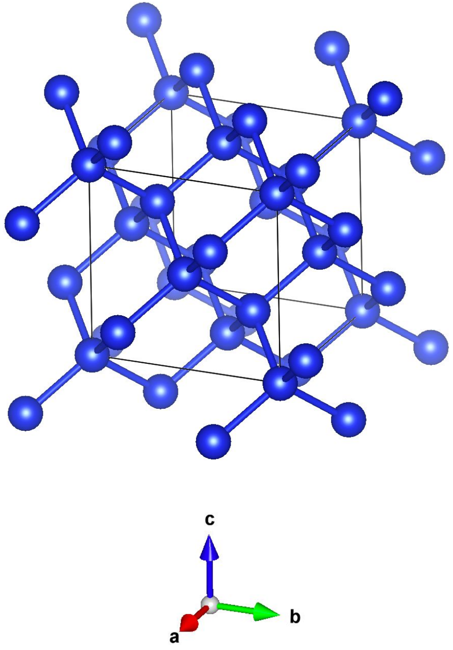
What are our data types in the physical sciences?
How do we build neural networks for these data types?

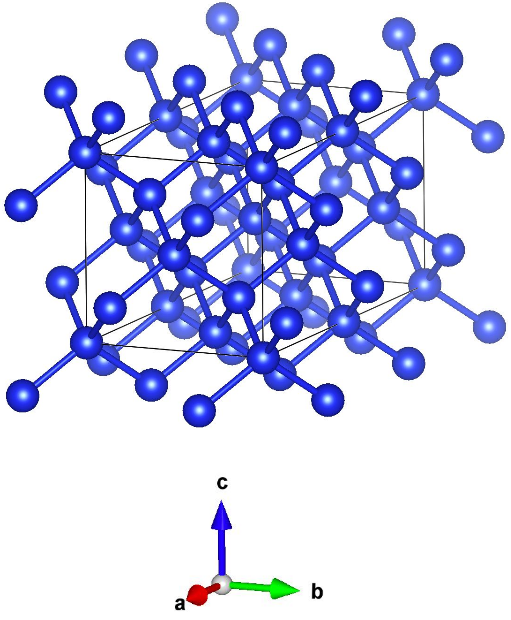
Given a molecule and a rotated copy,
we want the predicted forces to be the same up to rotation.
(Predicted forces are equivariant to rotation.)
Additionally, we should be able to generalize to molecules with similar motifs.



Primitive unit cells, conventional unit cells, and supercells of the same crystal should produce the same output (assuming periodic boundary conditions)**.**

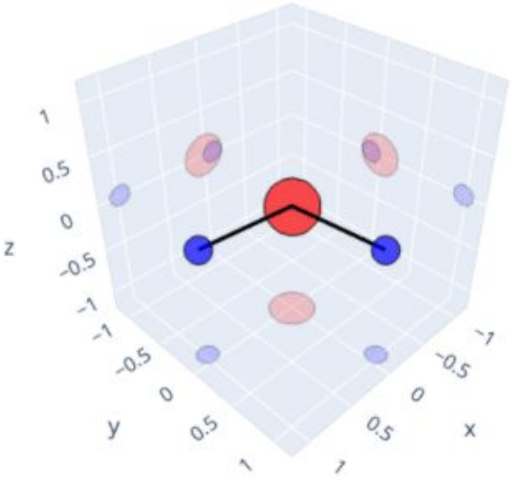




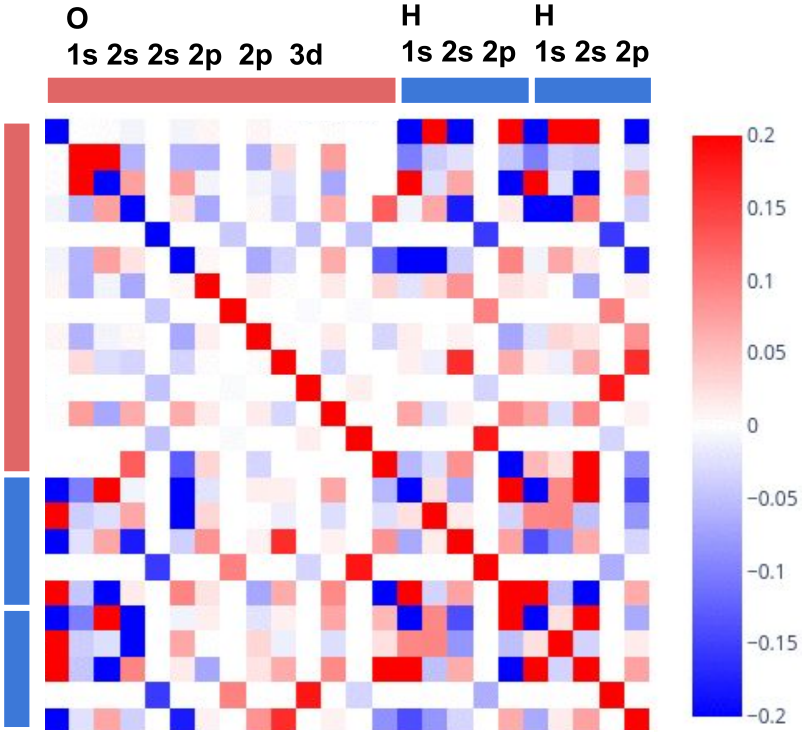


We want the networks to be able to predict molecular Hamiltonians in any orientation from seeing a single example.

$$R\vec{v}$$



$$RHR^T$$



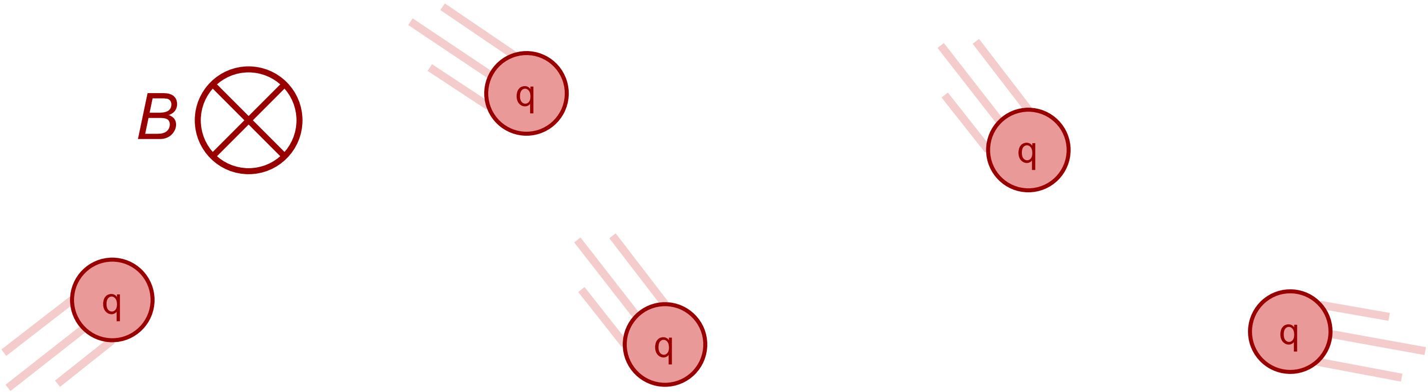
What our our data types?
3D geometry and geometric tensors...
...which transform predictably under 3D rotation, translation, and inversion.

These data types <u>assume</u> Euclidean symmetry.
⇨ Thus, we need neural networks that preserve Euclidean symmetry.

Analogous to... **the laws of** *(non-relativistic)* **physics have Euclidean symmetry,** ***even if systems do not.***

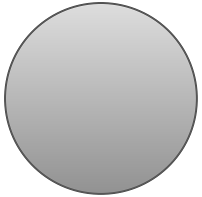
The **network** is our model of **"physics"**. The **input** to the network is our **system**.

$$\vec{F}(q, \vec{r}, \vec{v}, \vec{B}) = \vec{F}_i = \sum_i q_i(\vec{v}_i \times \vec{B}) + \sum_{i \neq j} \frac{q_i q_j}{r_{ij}^2} \hat{r_{ij}}$$



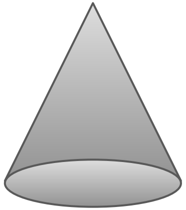
A Euclidean symmetry preserving network produces *outputs* that preserve the subset of symmetries induced by the *input*.

3D rotations and inversions



O(3)

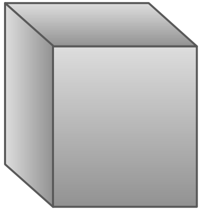
2D rotation and mirrors along cone axis



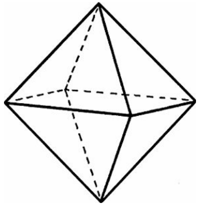
SO(2) + mirrors

($C_{\infty v}$)

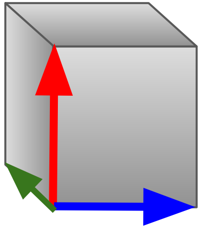
Discrete rotations and mirrors



O_h



Discrete rotations, mirrors, and translations



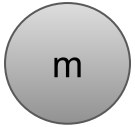
Pm-3m (221)

Geometric tensors take many forms. *They are a general data type beyond materials.*

Geometric tensors take many forms. *They are a general data type beyond materials.*

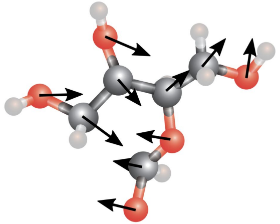
Scalars
- Energy
- Mass
- Isotropic *

$$E$$



Vectors
- Force
- Velocity
- Acceleration
- Polarization

$$\vec{F}$$

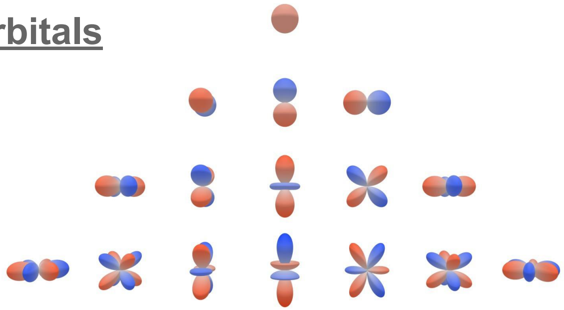


Pseudovectors
- Angular momentum
- Magnetic fields

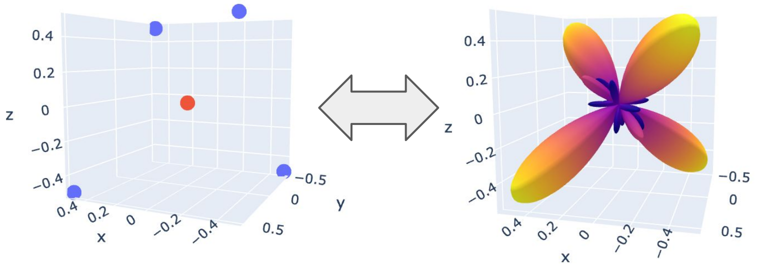
Matrices, Tensors, …
- Moment of Inertia
- Polarizability
- Interaction of multipoles
- Elasticity tensor (rank 4)

$$\alpha = \begin{bmatrix} \alpha_{xx} & \alpha_{xy} & \alpha_{xz} \\ \alpha_{yx} & \alpha_{yy} & \alpha_{yz} \\ \alpha_{zx} & \alpha_{zy} & \alpha_{zz} \end{bmatrix}$$

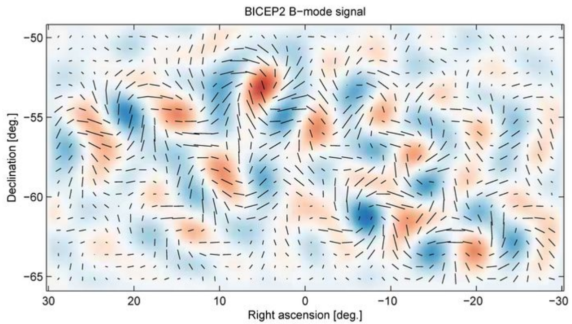
Atomic orbitals



Output of Angular Fourier Transforms



Vector fields on spheres
(e.g. B-modes of the Cosmic Microwave Background)



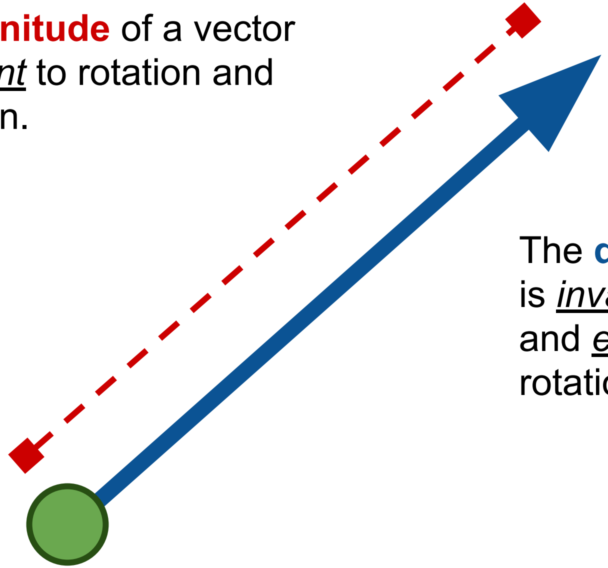
Geometric tensors only permit specific operations.

(More about these later -- scalar operations, direct sums, direct products)

Neural networks that only use these operations are *equivariant* to 3D translations, rotations, and inversion.

***Equivariant vs. Invariant?* Examples for a vector.**

The **magnitude** of a vector is *invariant* to rotation and translation.



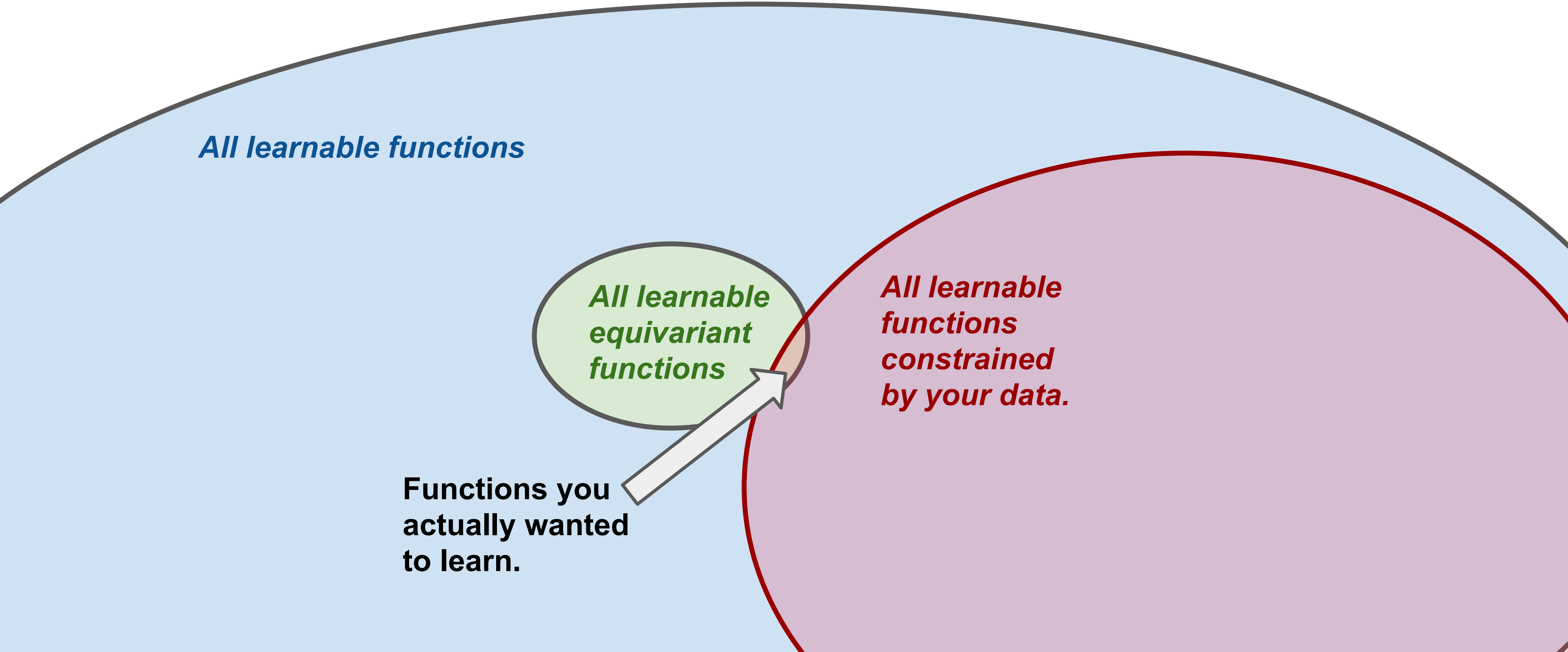
The **direction** of a vector is *invariant* to translation and *equivariant* to rotation.

The **location** of a vector in space is *equivariant* to translation and *equivariant* to rotation.

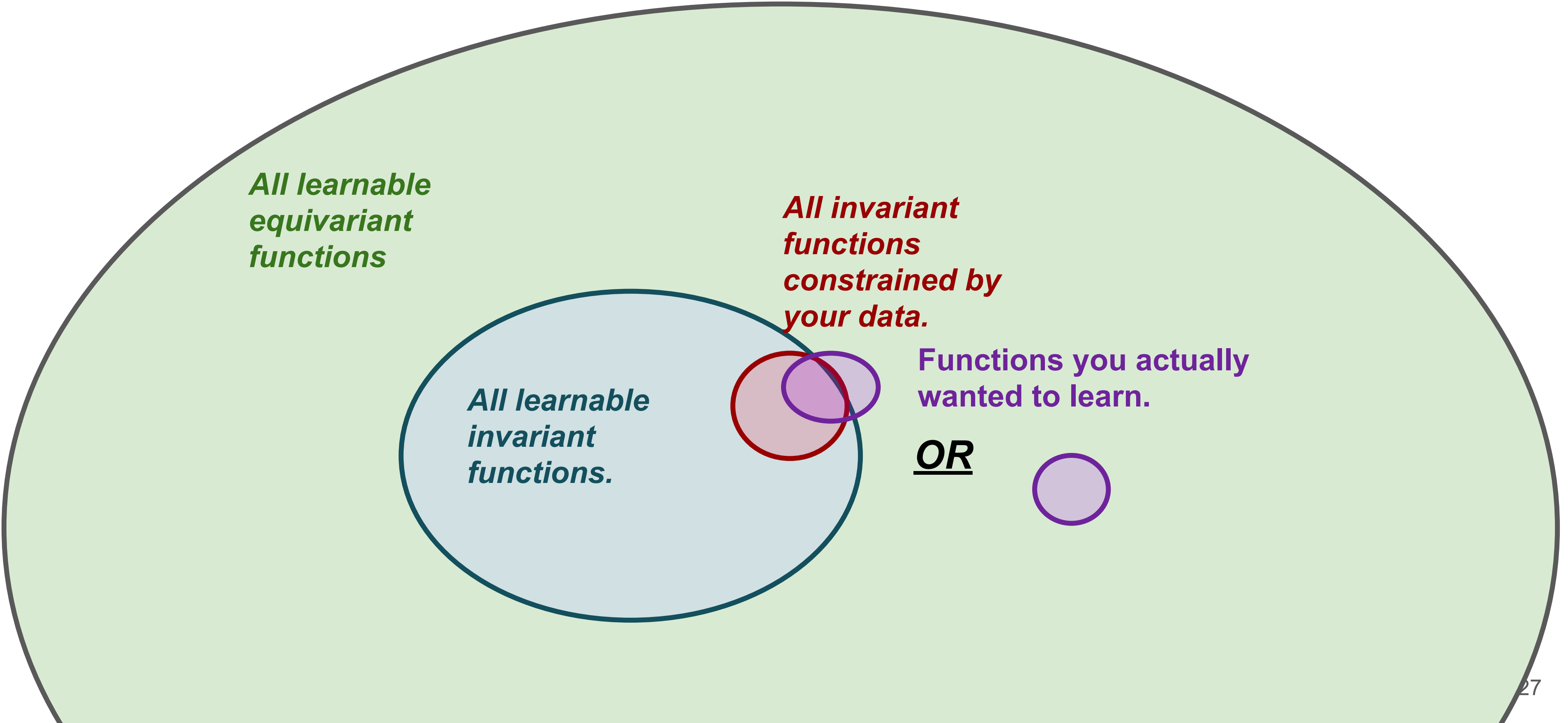
Why limit yourself to equivariant functions?

You can *substantially* shrink the space of functions you need to optimize over.

This means you need less data to constrain your function.

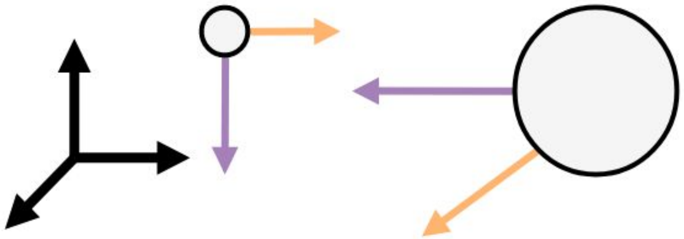


Why not limit yourself to invariant functions?
You have to <u>guarantee</u> that your input features already contain any necessary equivariant interactions ***(e.g. cross-products)*****.**



Building Euclidean Neural Networks

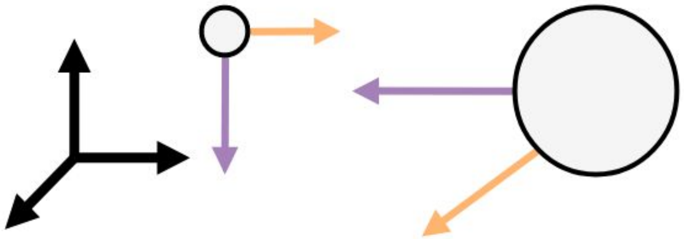
The input to our network is geometry and features on that geometry.



```
[[[m0]],[[m1]]],
[[[v0x, v0y, v0z],[a0x, a0y, a0z]],
 [[v1x, v1y, v1z],[a1x, a1y, a1z]]]
```

The input to our network is geometry and features on that geometry.
We categorize our features by how they transform under rotation.

Features have "angular frequency" L
where L is a positive integer.



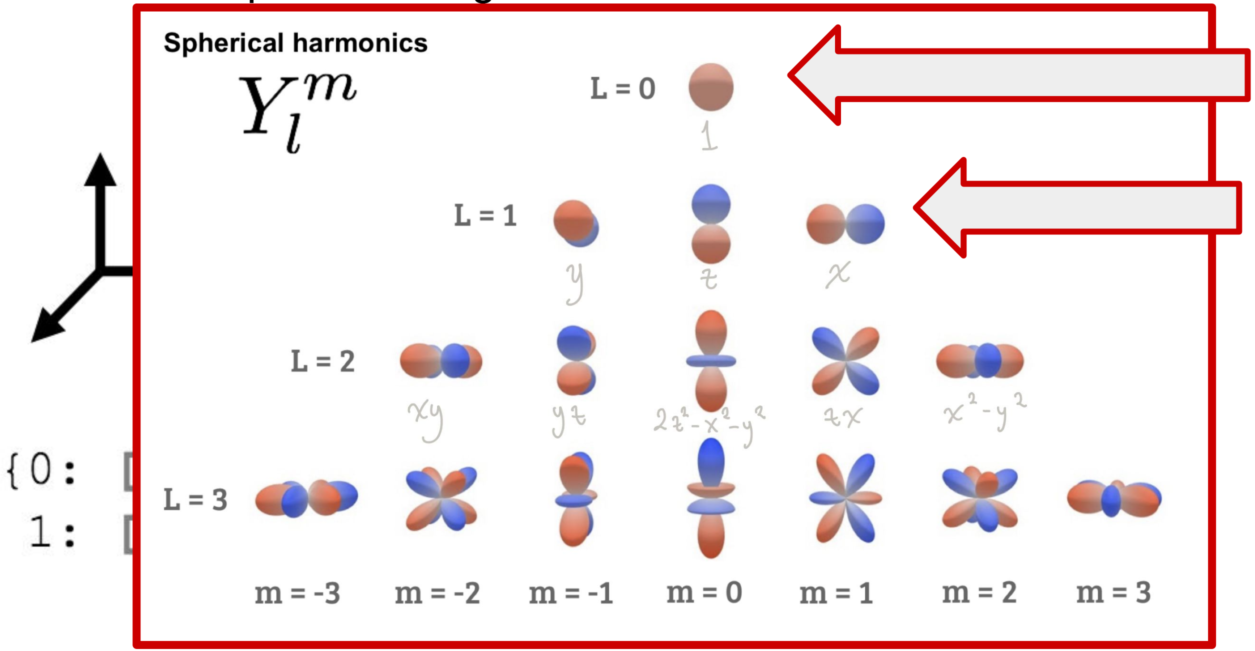
```
{0: [[[m0]],[[m1]]],
 1: [[[v0x, v0y, v0z],[a0x, a0y, a0z]],
     [[v1x, v1y, v1z],[a1x, a1y, a1z]]]}
```

	Frequency	
Scalars	$l = 0$	Doesn't change with rotation
Vectors	$l = 1$	Changes with same frequency as rotation
3x3 Matrices	$l = 0 \oplus 1 \oplus 2$	

The input to our network is geometry and features on that geometry.

We categorize our features by how they transform under rotation.

Features have “angular frequency” L
where L is a positive integer.



<u>Frequency</u>

$$l = 0$$

Doesn’t change with rotation

$$l = 1$$

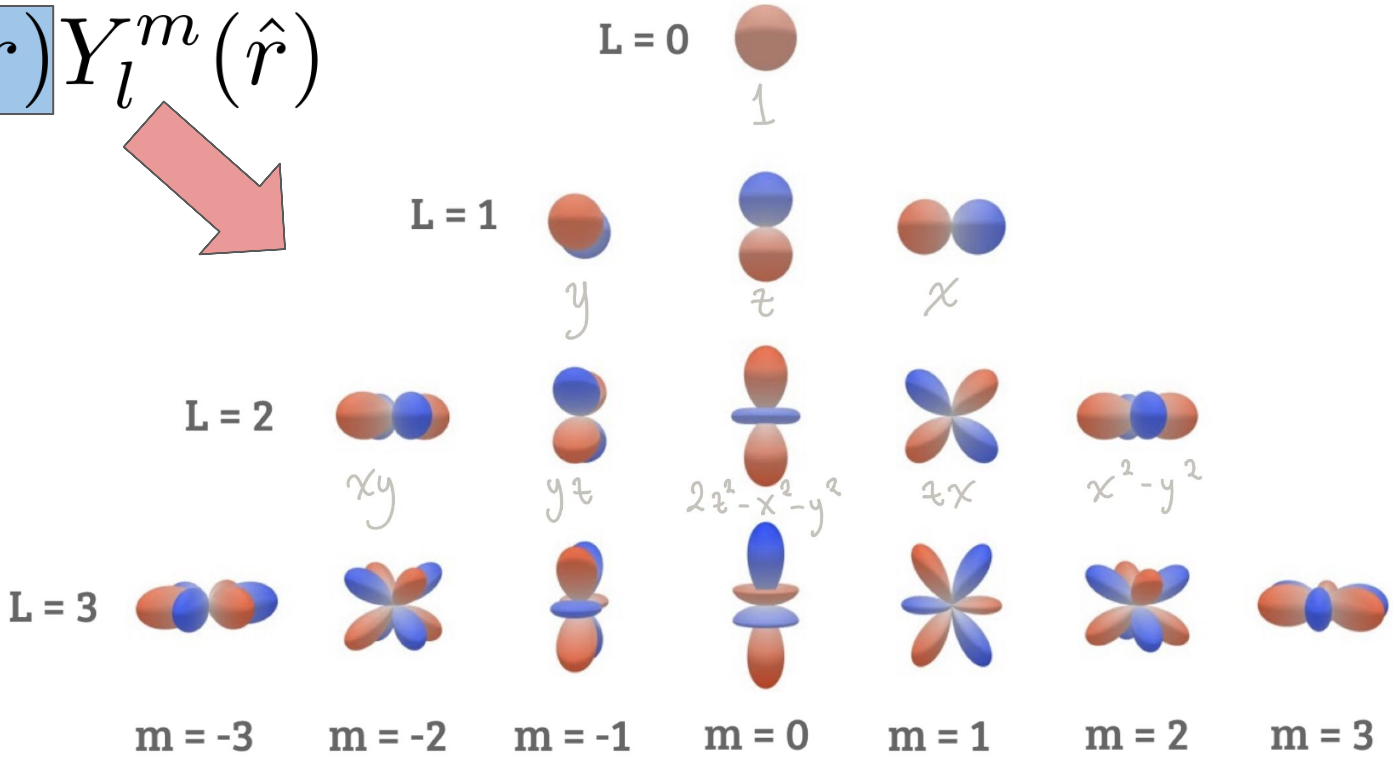
Changes with same frequency as rotation

$$l = 0 \oplus 1 \oplus 2$$

Euclidean Neural Networks are similar to convolutional neural networks,
EXCEPT with special filters and tensor algebra!

Convolutional filters based on learned radial functions and spherical harmonics.

$$W(\vec{r}) = \boxed{R(r)}\, Y_l^m(\hat{r})$$



Euclidean Neural Networks are similar to convolutional neural networks,
EXCEPT with special filters and tensor algebra!

Everything in the network is a geometric tensor!

Scalar multiplication gets replaced with the more general tensor product. $\bigotimes$

Contract two indices to one with Clebsch-Gordan Coefficients.

Example: How do you "multiply" two vectors?

Dot product

$$\begin{pmatrix} a_i & a_j & a_k \end{pmatrix} \begin{pmatrix} b_i \\ b_j \\ b_k \end{pmatrix} = c$$

Scalar, Rank-0

Cross product

$$\vec{a} \times \vec{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_i & a_j & a_k \\ b_i & b_j & b_k \end{vmatrix} = \vec{c}$$

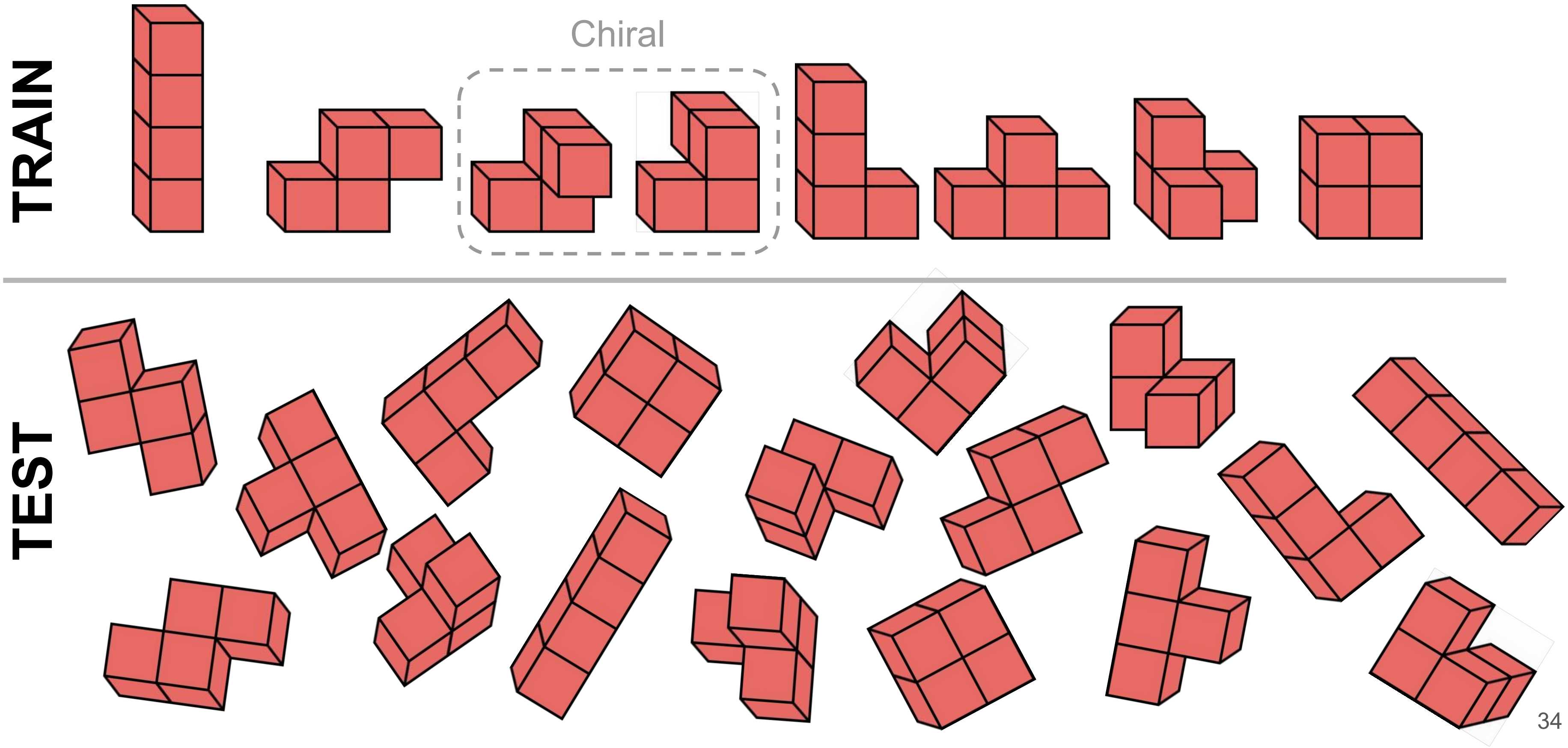
Vector, Rank-1

Matrix, Rank-2

Outer product

$$\begin{pmatrix} a_i \\ a_j \\ a_k \end{pmatrix} \begin{pmatrix} b_i & b_j & b_k \end{pmatrix} = \begin{pmatrix} a_i b_i & a_i b_j & a_i b_k \\ a_j b_i & a_j b_j & a_j b_k \\ a_k b_i & a_k b_j & a_k b_k \end{pmatrix}$$

Our unit test: Trained on 3D Tetris shapes in one orientation, these network can perfectly identify these shapes in any orientation.



Applications

Laundry list…

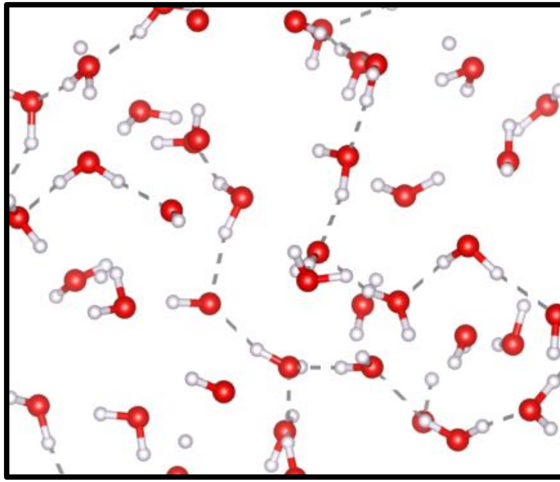
- Inverting invariant representations
- Molecular dynamics
- Autoencoder for Geometry
- Determining missing data input through symmetry
- Electron density prediction for large molecules
- Molecule and crystal property prediction
- Conditional protein design
- ...

Predict ab initio forces for molecular dynamics

Preliminary results originally presented at
APS March Meeting 2019.
Paper in progress.

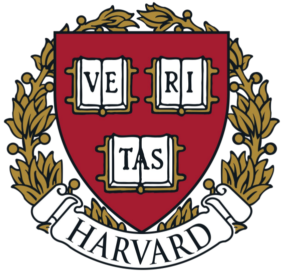
Testing on liquid water, Euclidean neural networks *(Tensor-Field Molecular Dynamics)* **require less data to train than traditional networks to get state of the art results.**

	MAE [meV/A]	RMSE [meV/A]
TFMD, 100	27.9	38.20
TFMD, 1000	**11.29**	**14.82**
Deep-MD, 133,500	not reported	40.0



Simon Batzner

Boris Kozinsky



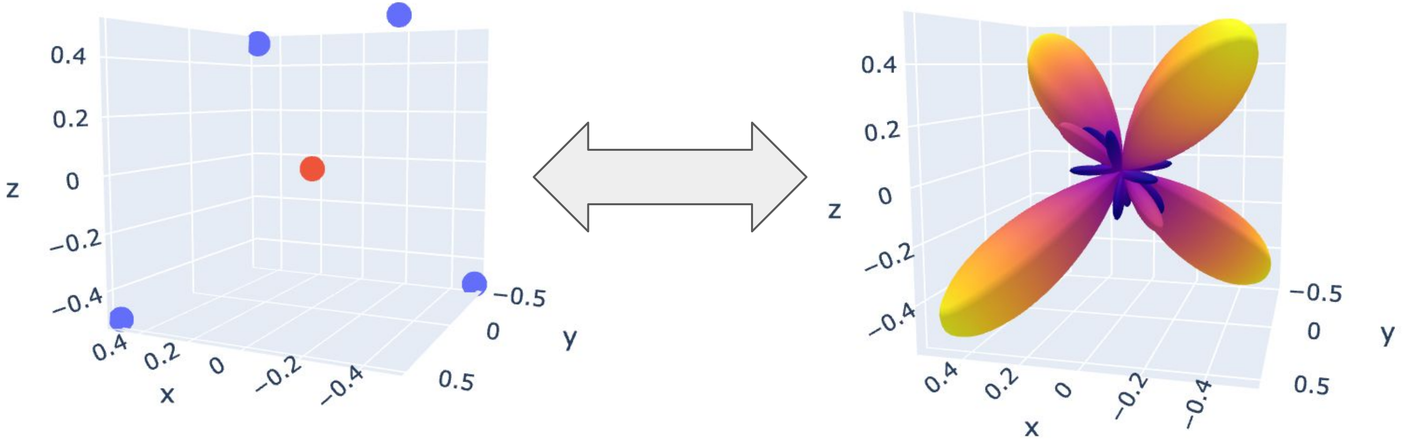
Data set from: [1]
Zhang, L. et al. E. (2018).
PRL, *120*(14), 143001.

Euclidean neural networks can manipulate geometry,
which means they can be used for generative models such as autoencoders.

Euclidean neural networks can manipulate geometry,
which means they can be used for generative models such as autoencoders.

To encode/decode, we have to be able to convert *geometry* into *features* and *vice versa*. We do this via spherical harmonic projections.

geometry ***features***

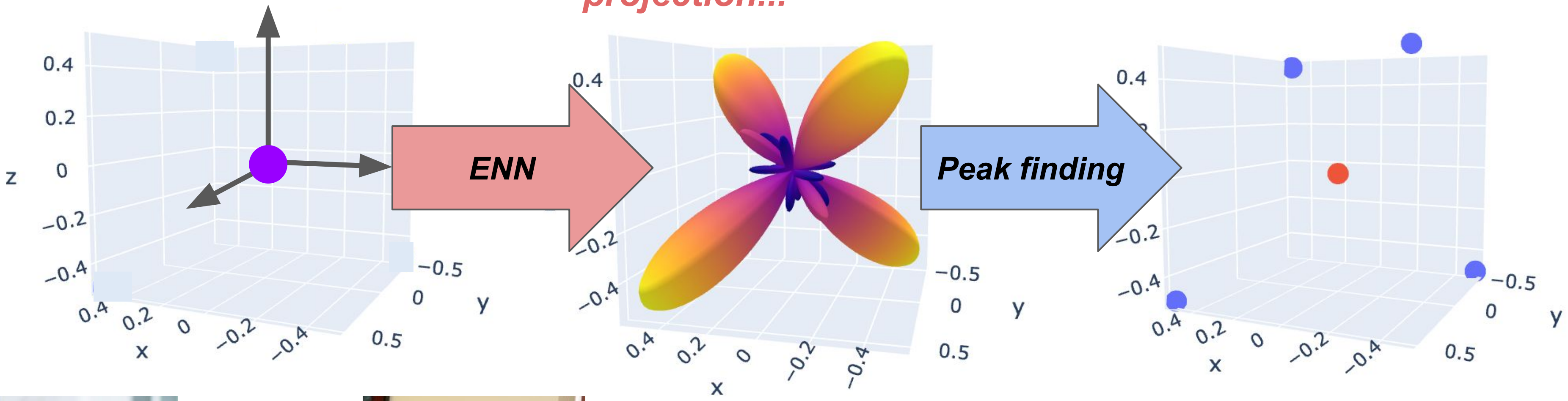


Equivariant neural networks can learn to invert invariant representations.

Invariant features + coordinate frame

Network can predict spherical harmonic projection...

Which can be used to recover geometry.

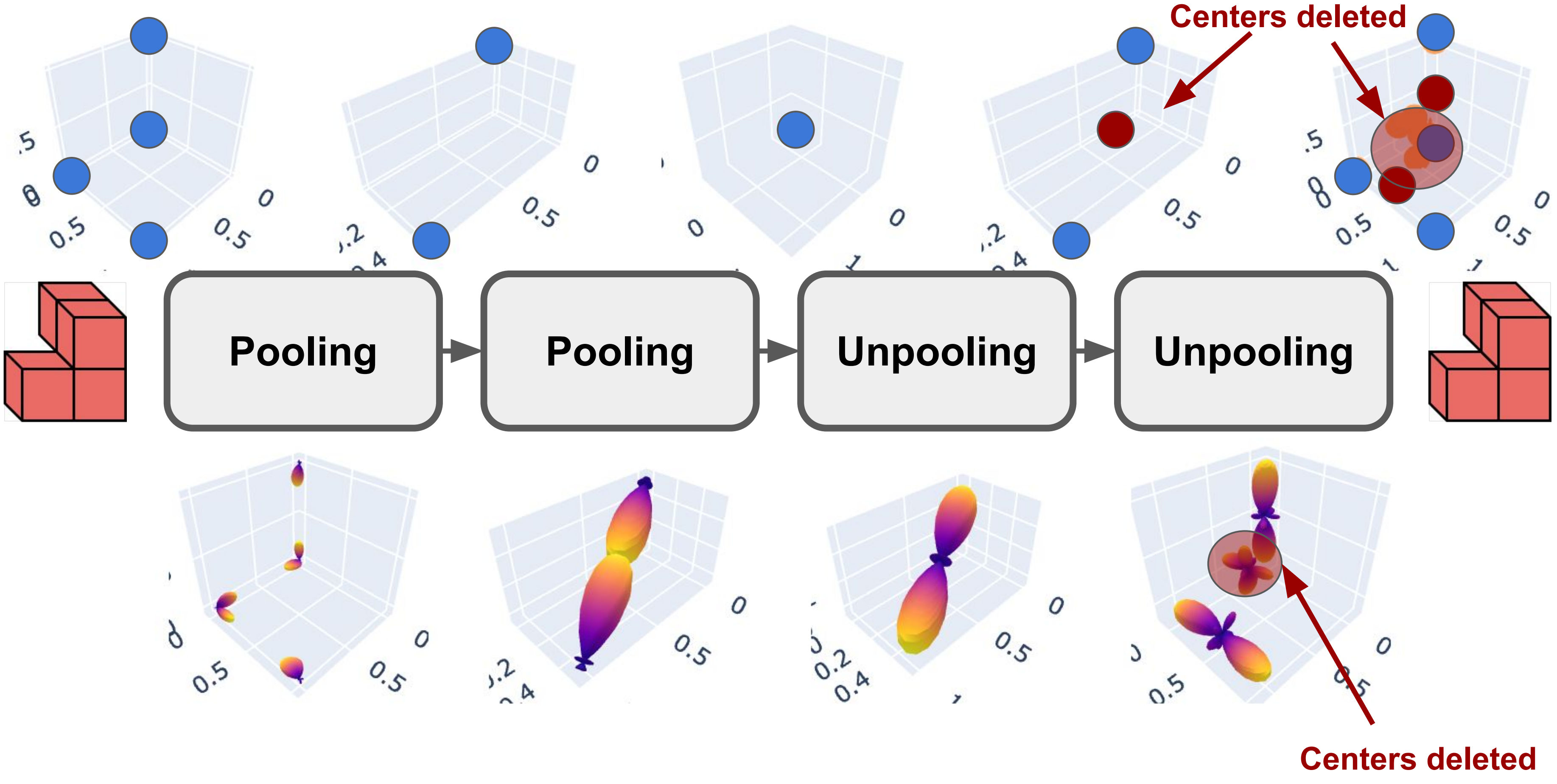


Thomas
Hardin

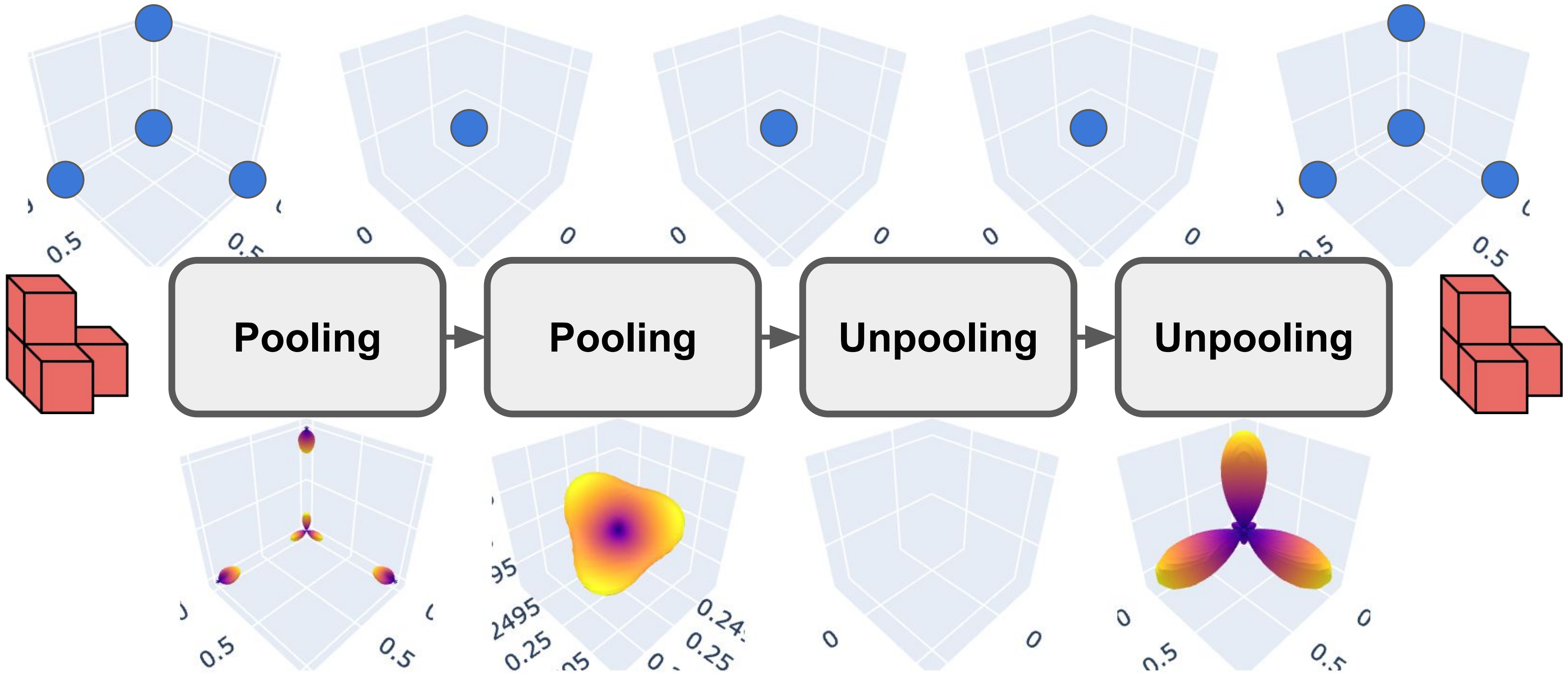
Josh
Rackers

Sandia
National
Laboratories

We can also build an autoencoder for geometry: e.g. Autoencoder on 3D Tetris



We can also build an autoencoder for geometry: e.g. Autoencoder on 3D Tetris



developers of `e3nn`
(and *atomic architects*)

Mario Geiger

Ben Miller

Koctiantyn Lapchevskyi

Tess Smidt

tensor field networks

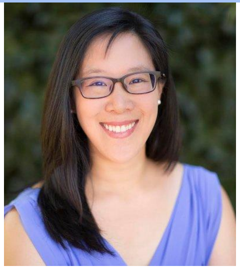
Stanford

Nate Thomas

Google Accelerated Science Team

Patrick Riley

Steve Kearnes



Lusann Yang

Li Li

Kai Kohlhoff

Euclidean neural networks operate on points/voxels and have symmetries of E(3).

- The inputs and outputs of our network are geometry and geometric tensors.
- Convolutional filters are built from spherical harmonics with a learned radial function.
- All network operations are compatible with geometric tensor algebra.

We expect these networks to be generally useful for physics, chemistry, and geometry.

So far these networks have learned efficient molecular dynamics models and can learn to recursively encode and decode geometry.

Reach out to me if you are interested and/or have any questions!

`e3nn` Code (PyTorch):
http://github.com/e3nn/e3nn

`e3nn_tutorial`
https://blondegeek.github.io/e3nn_tutorial/

Tensor Field Networks
(arXiv:1802.08219)

3D Steerable CNNs
(arXiv:1807.02547)

Tess Smidt
tsmidt@lbl.gov

Calling in backup (slides)!

Several groups converged on similar ideas around the same time.

Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds
(arXiv:1802.08219)
Tess Smidt*, Nathaniel Thomas*, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, Patrick Riley

Points, nonlinearity on norm of tensors

Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network
(arXiv:1806.09231)
Risi Kondor, Zhen Lin, Shubhendu Trivedi

Only use tensor product as nonlinearity, no radial function

3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data
(arXiv:1807.02547)
Mario Geiger*, Maurice Weiler*, Max Welling, Wouter Boomsma, Taco Cohen

Efficient framework for voxels, gated nonlinearity

**denotes equal contribution*

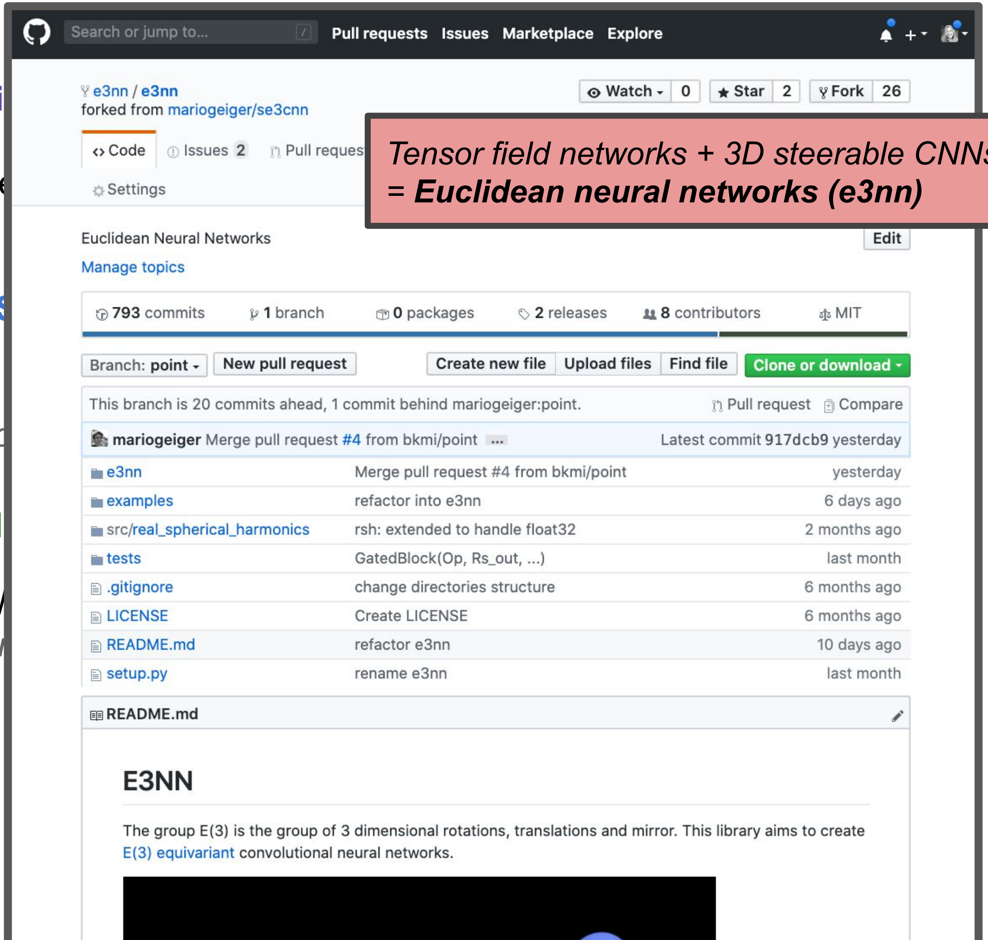
Several groups converged on similar ideas around the same time.

Tensor field networks: Rotation- and translati
(arXiv:1802.08219)
Tess Smidt*, Nathaniel Thomas*, Steven Kearne
Points, nonlinearity on norm of tensors

Clebsch-Gordan Nets: a Fully Fourier Space S
(arXiv:1806.09231)
Risi Kondor, Zhen Lin, Shubhendu Trivedi
Only use tensor product as nonlinearity, no

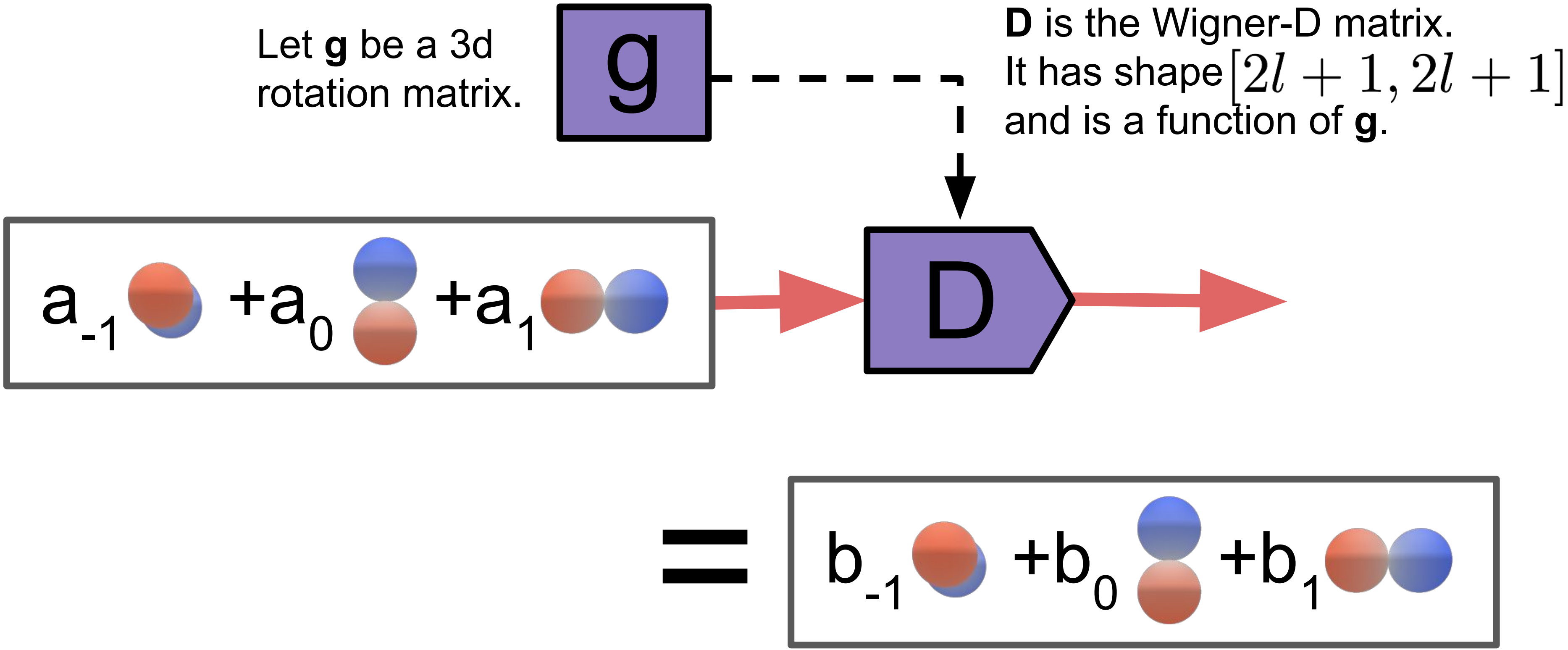
3D Steerable CNNs: Learning Rotationally Eq
(arXiv:1807.02547)
Mario Geiger*, Maurice Weiler*, Max Welling, W
Efficient framework for voxels, gated nonli

**denotes equal contribution*

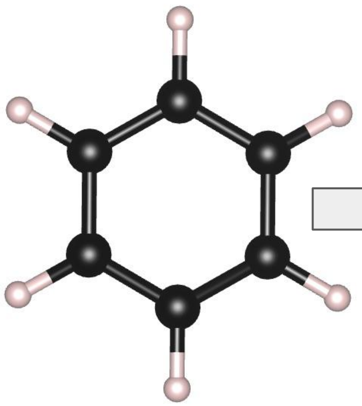


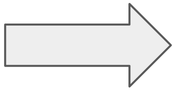
Tensor field networks + 3D steerable CNNs
*= **Euclidean neural networks (e3nn)***

Spherical harmonics of a given L transform together under rotation.



How do we represent geometric data with neural networks (inputs / outputs)?





H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630
C	-1.78983	-0.36233	-2.36935
H	-2.72799	-0.85413	-2.64566
C	-0.81200	-0.13809	-3.33310
H	-0.98066	-0.45335	-4.36774
C	0.38026	0.48673	-2.98192
H	1.14976	0.66307	-3.74025
C	0.59460	0.88737	-1.66708
H	1.53276	1.37906	-1.39070

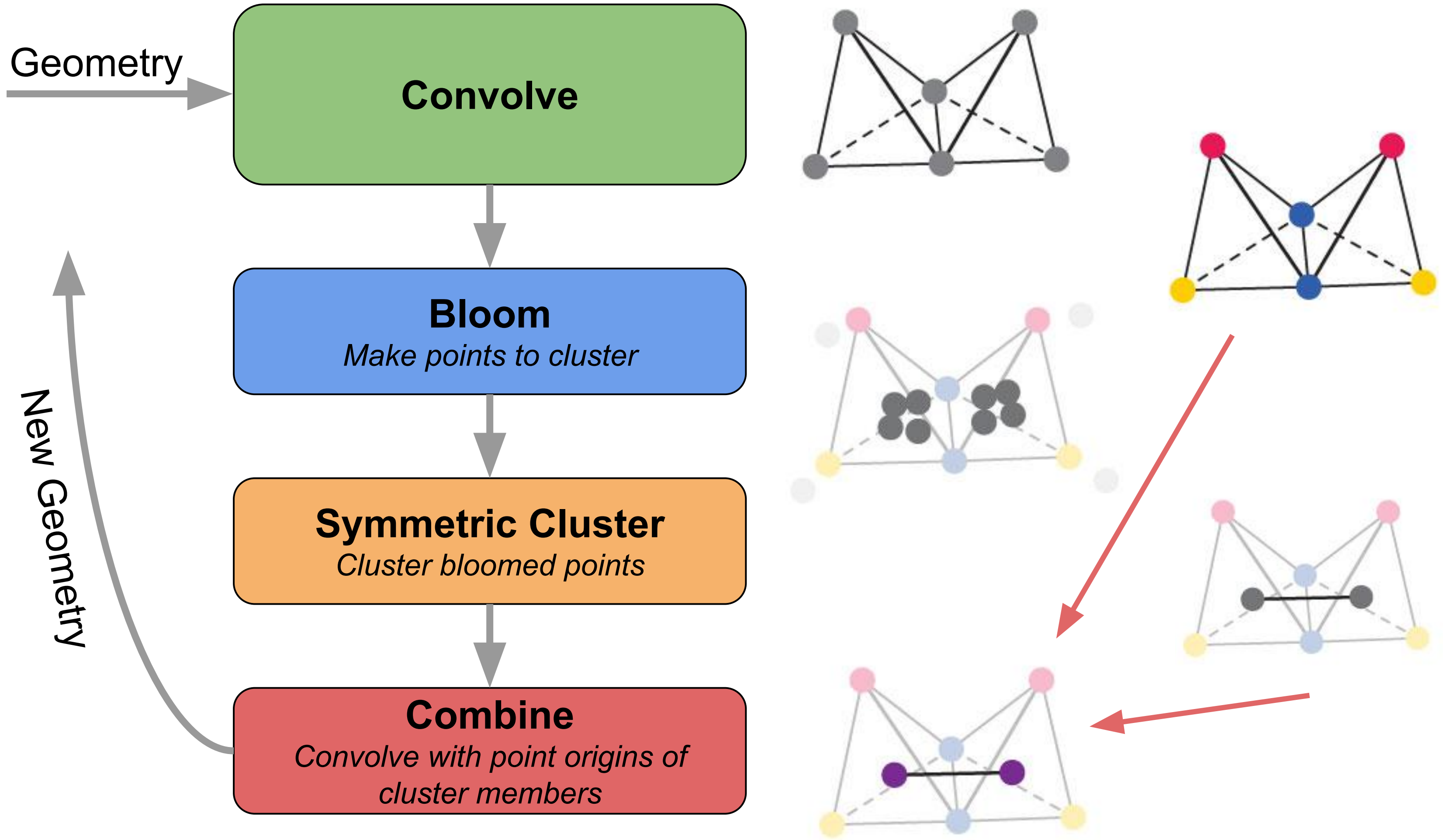
Coordinates are most general, but sensitive to translations and rotations.

Approach 1:
It doesn't matter! It's deep learning! Throw all your data at the problem and see what you get!

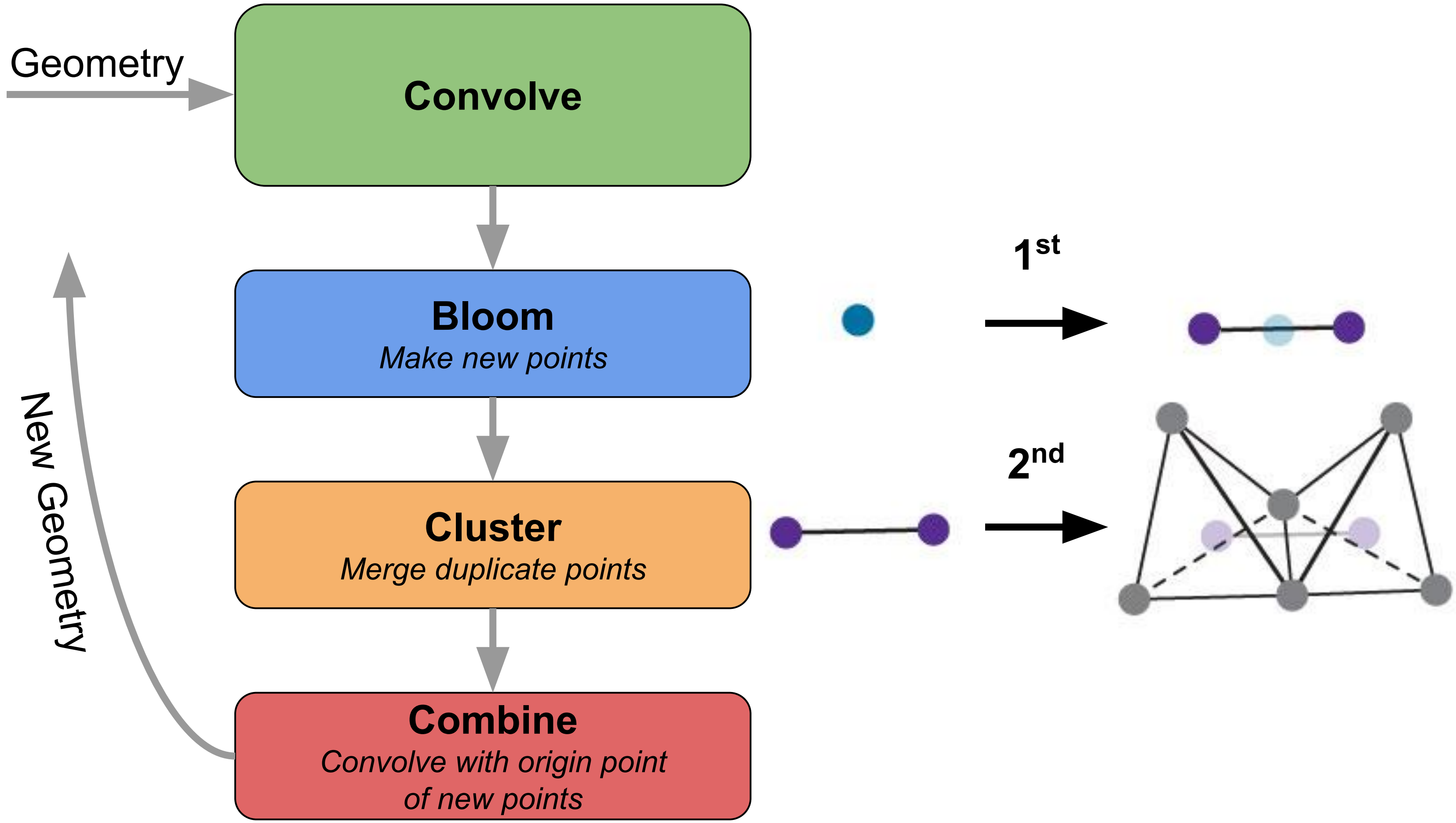
Approach 2: 
Convert your data to invariant representations so the neural network can't possibly mess it up.

Approach 3:
If there's no model that naturally handles coordinates, we will make one.

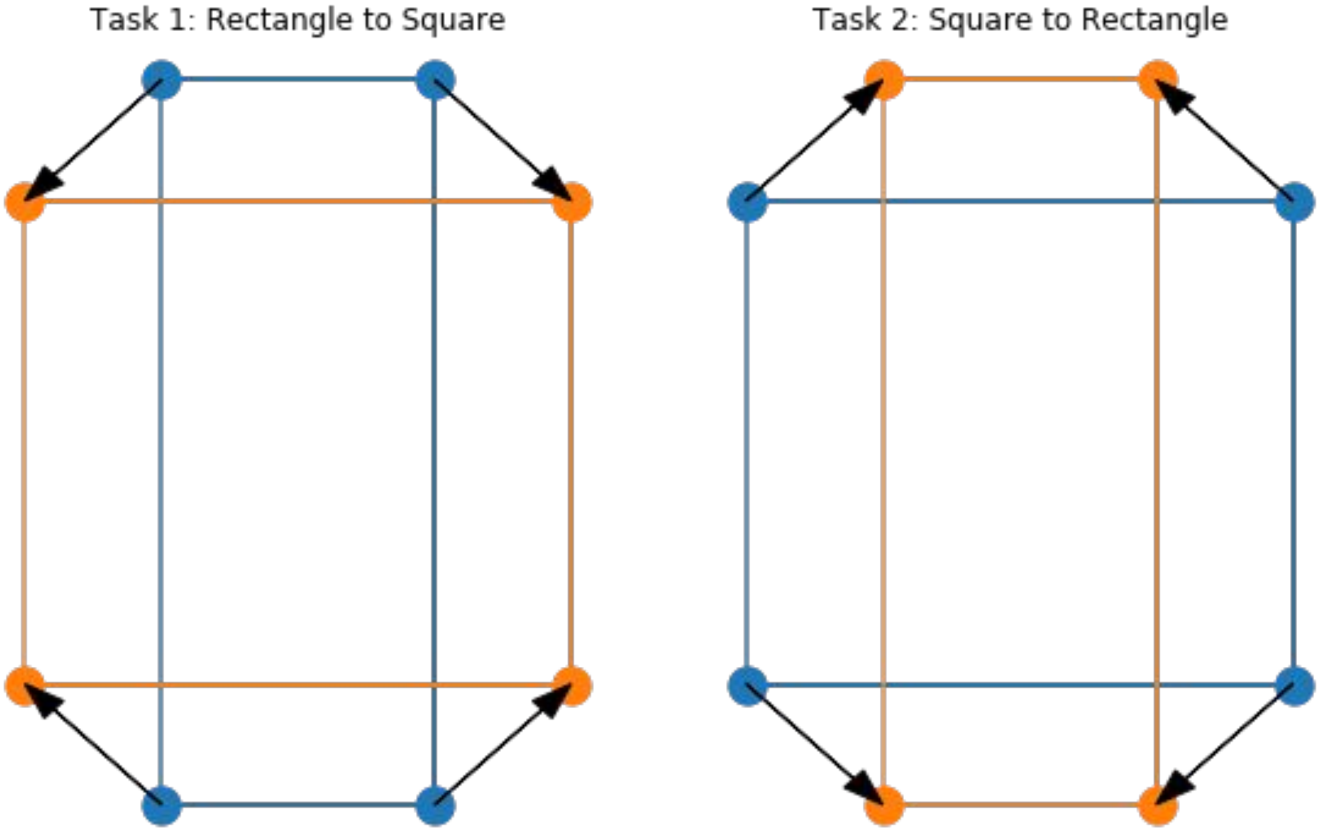
How to encode *(Pooling layer).* **Recursively convert geometry to features.**



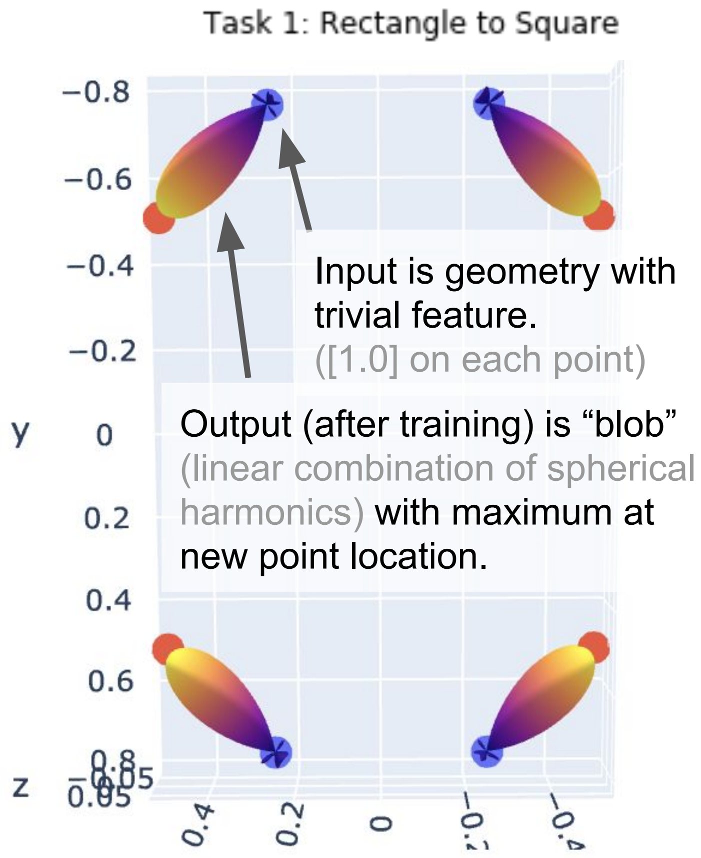
How to decode ***(Unpooling layer).*** **Recursively convert features to geometry.**



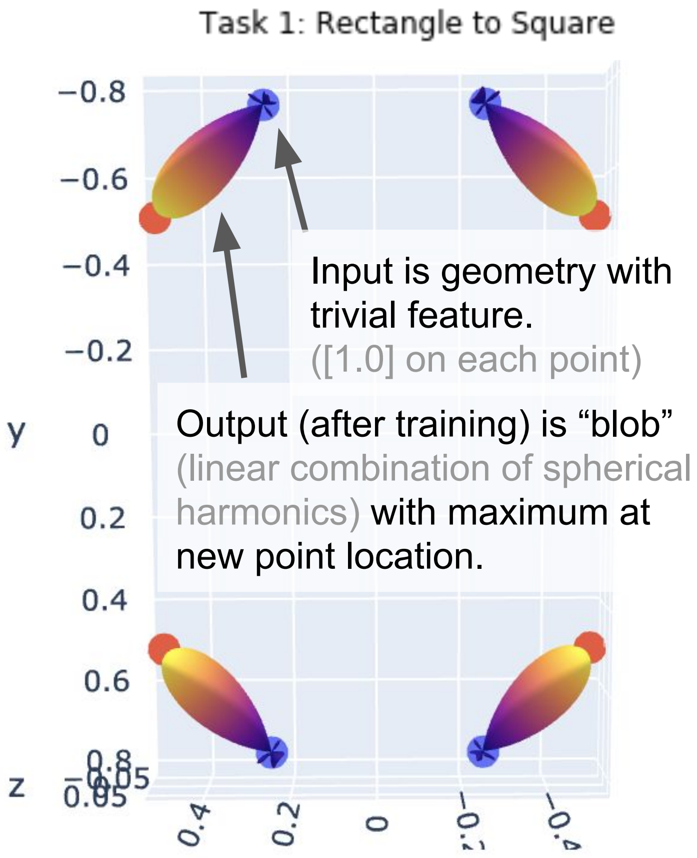
The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.

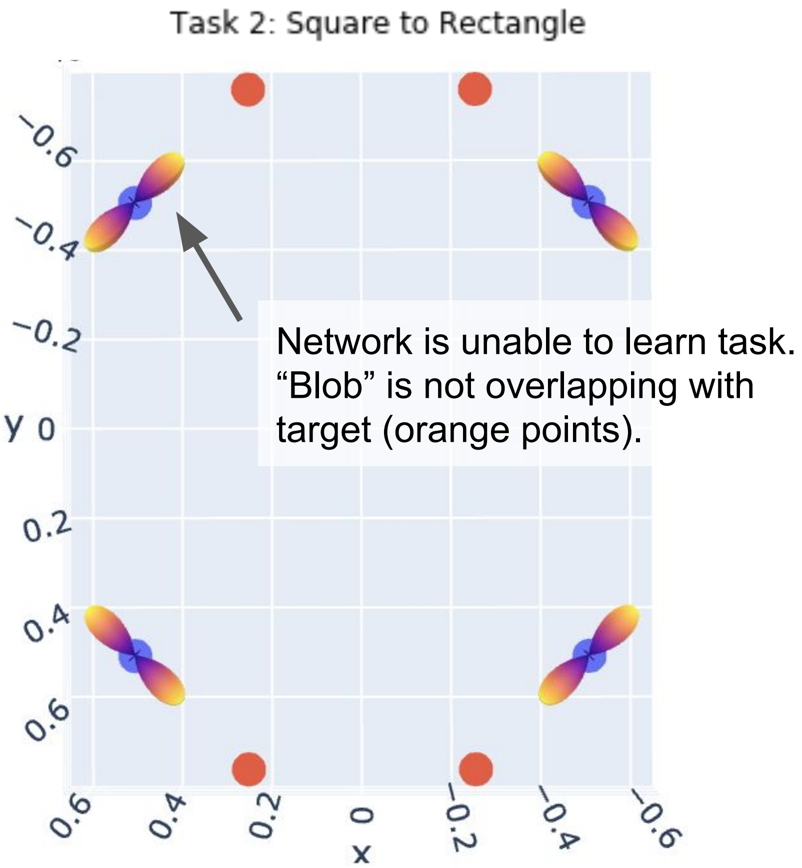


The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.



The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.

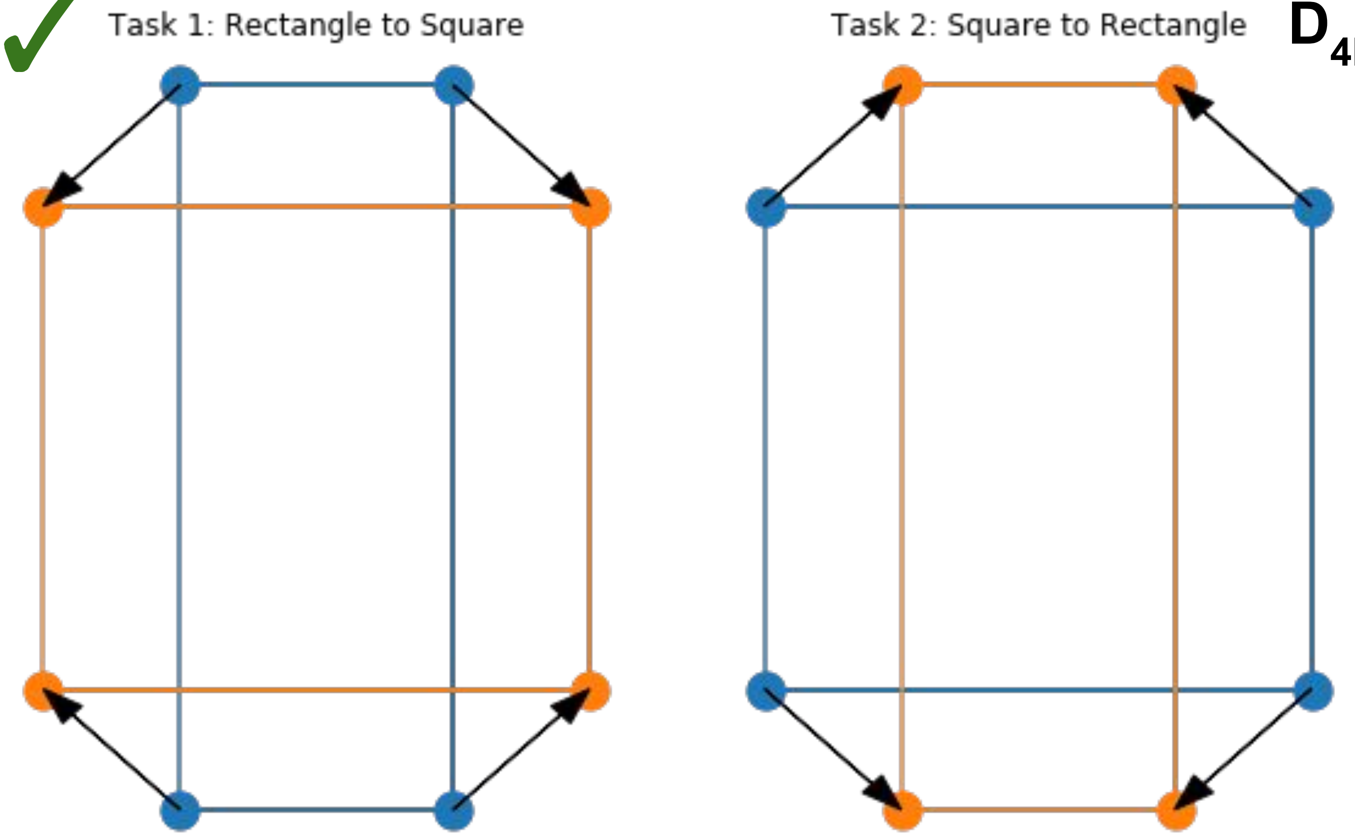




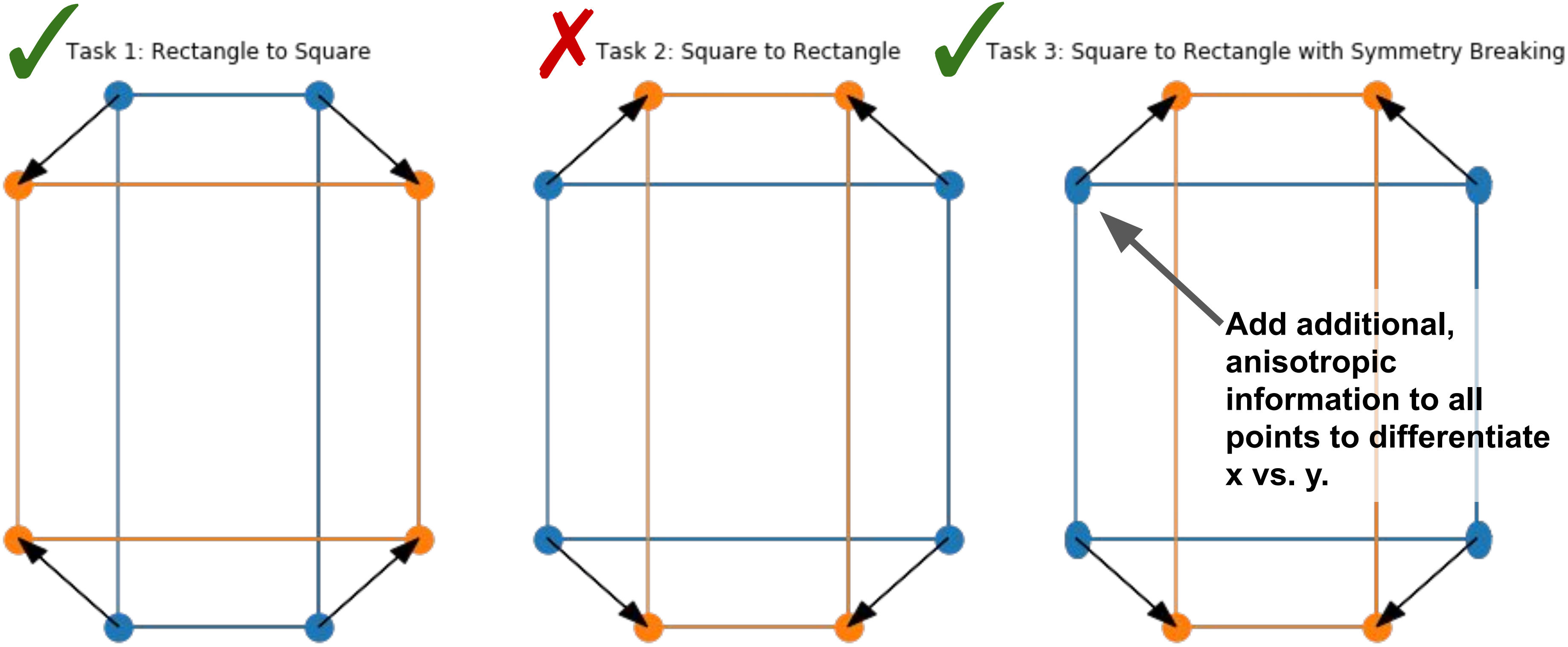
The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.

$D_{2h} \rightarrow D_{4h}$ ✓

$D_{4h} \rightarrow D_{2h}$ ✗

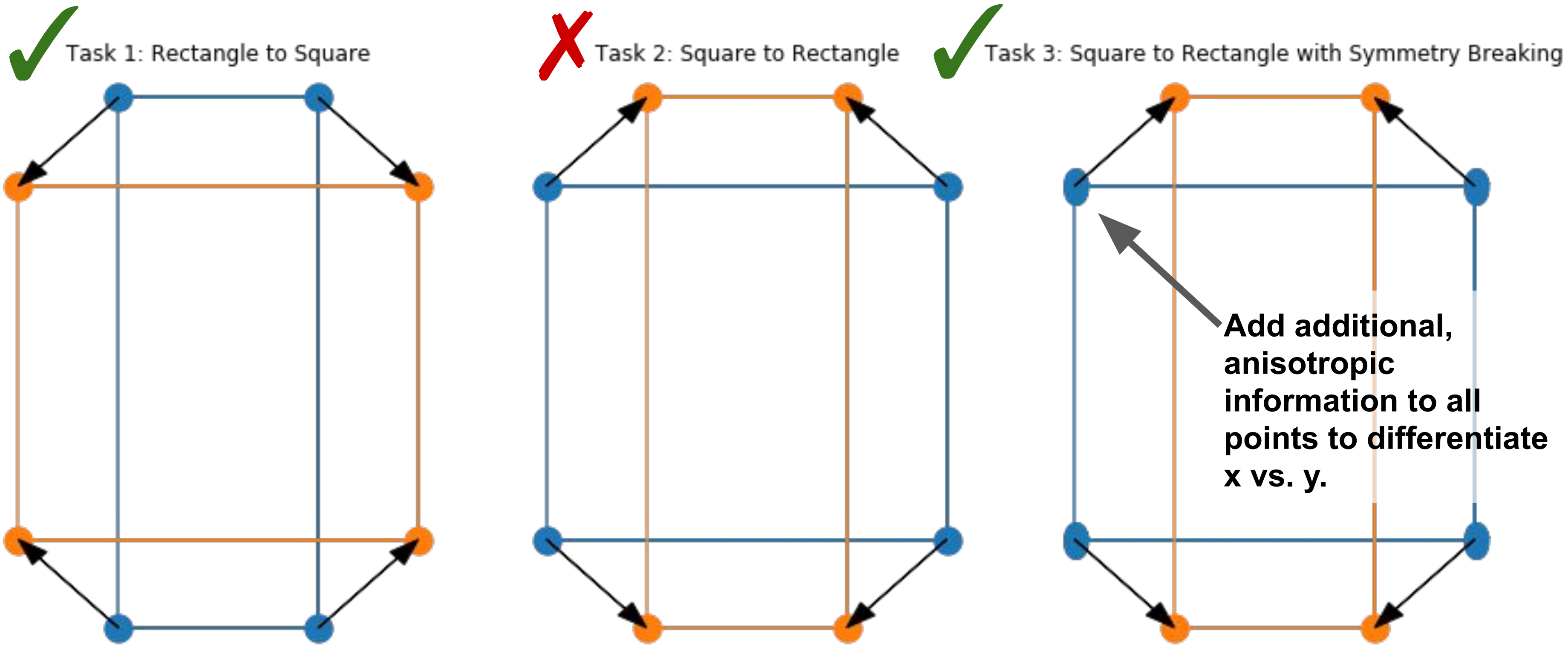


The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.



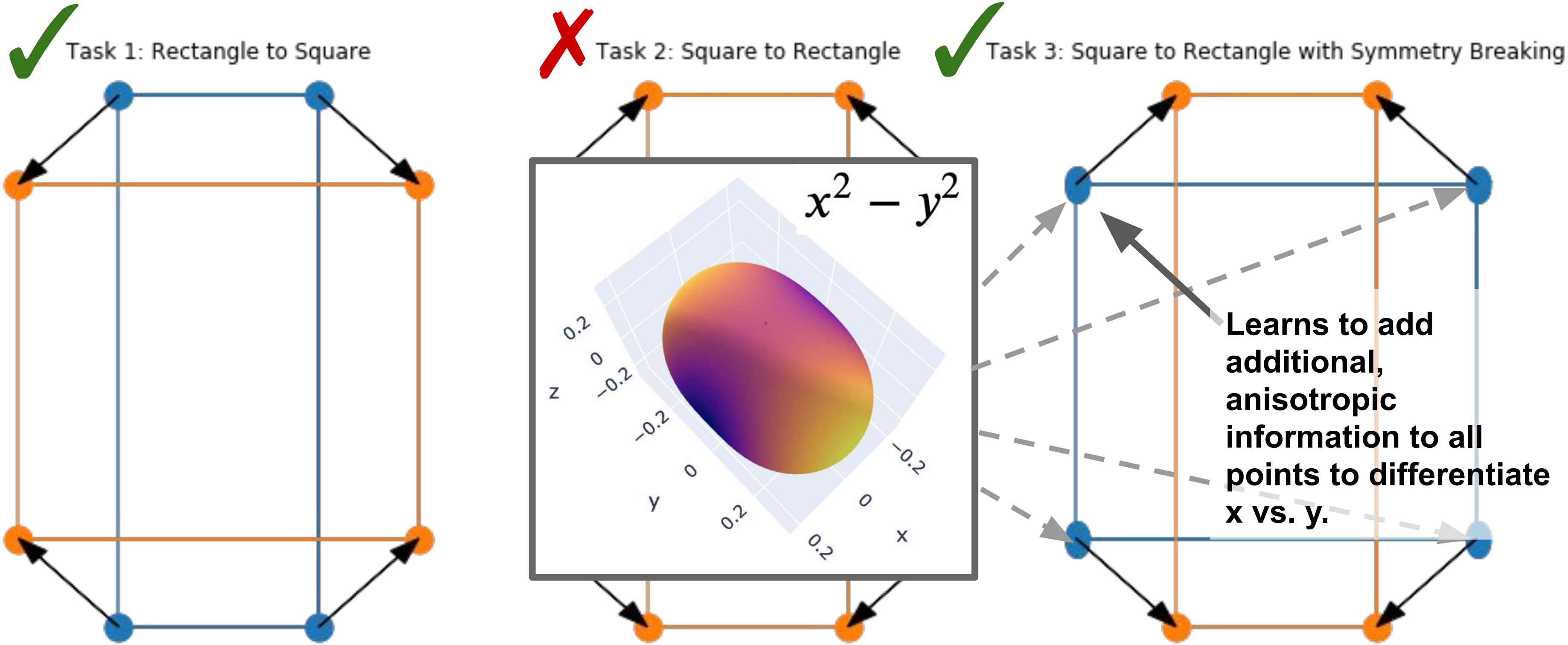
https://blondegeek.github.io/e3nn_tutorial/simple_tasks_and_symmetry.html

The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.



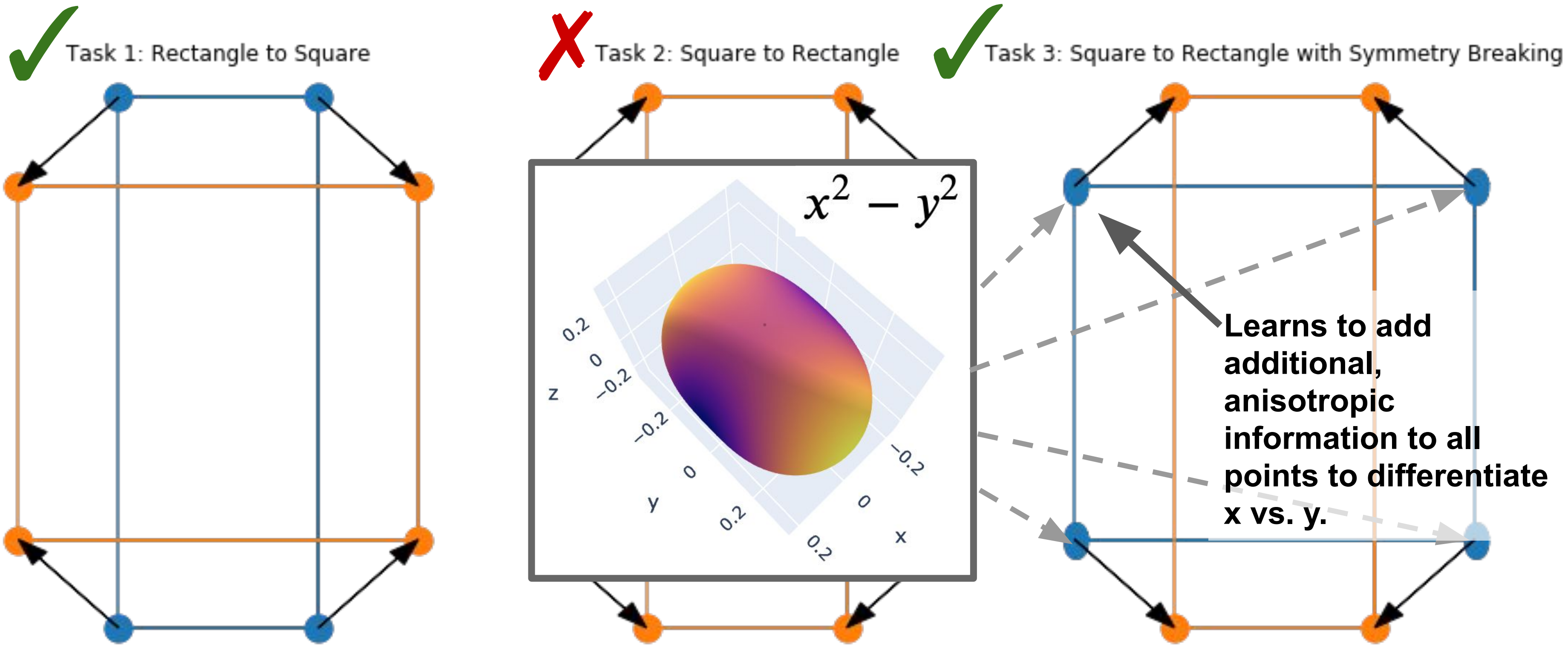
https://blondegeek.github.io/e3nn_tutorial/simple_tasks_and_symmetry.html

The outputs of the network must have equal or higher symmetry than the inputs.



https://blondegeek.github.io/e3nn_tutorial/simple_tasks_and_symmetry.html

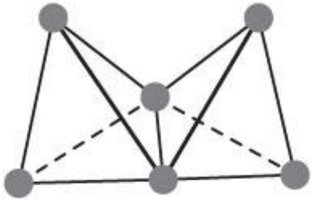
The outputs of the network must have equal or <u>higher</u> symmetry than the inputs.



Physics plays by the same rules! Physical processes must choose from energetically degenerate options to "break symmetry".

https://blondegeek.github.io/e3nn_tutorial/simple_tasks_and_symmetry.html

We want to convert geometric information (3D coordinates of atomic positions) into features on a trivial geometry (a single point) and back again.



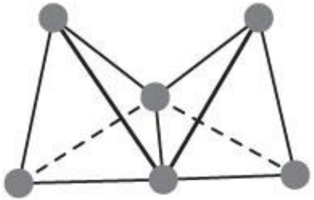
Discrete geometry

Reduce geometry to single point.

$\mathbb{R}^N$

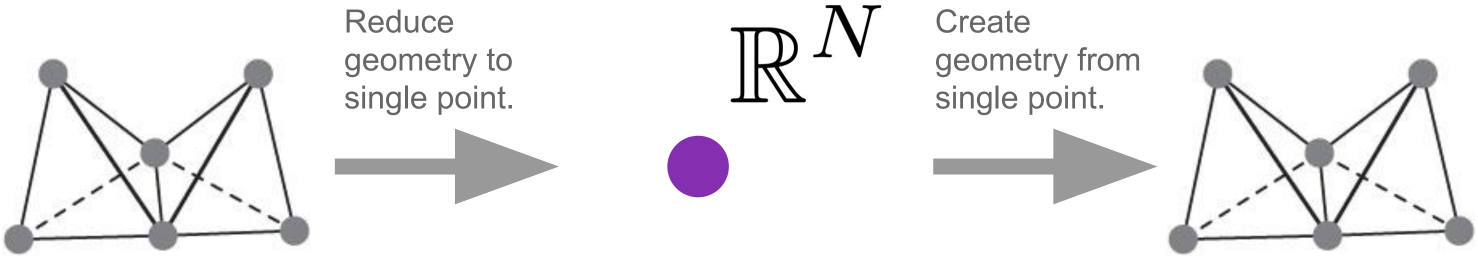
Single point with continuous latent representation (N dimensional vector)

Create geometry from single point.



Discrete geometry

We want to convert geometric information (3D coordinates of atomic positions) **into features on a trivial geometry** (a single point) **and back again.**

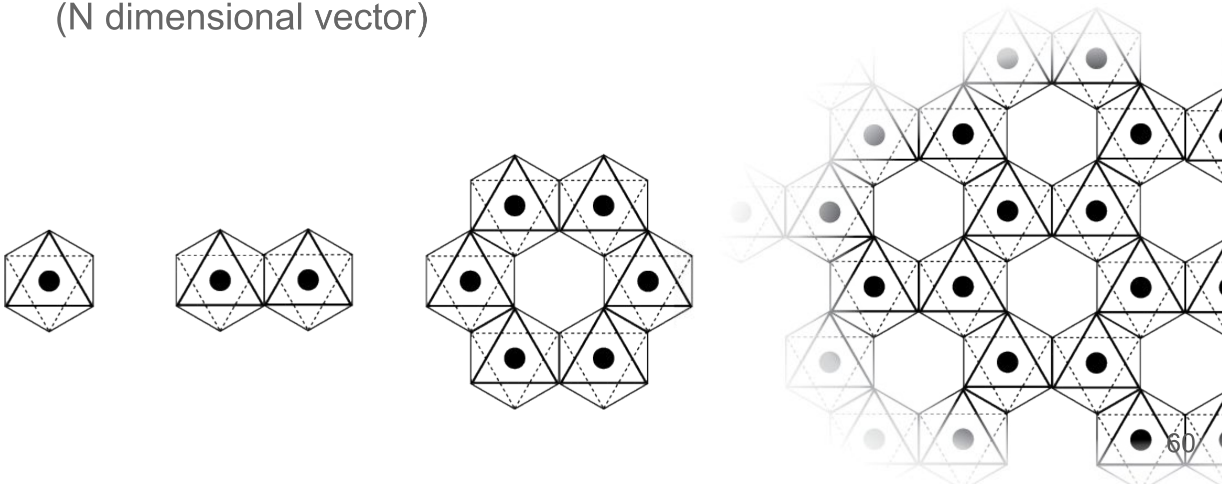


Discrete geometry

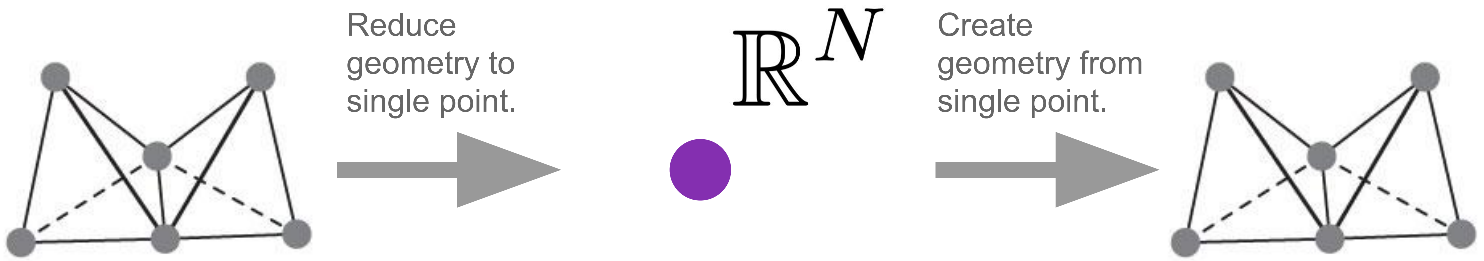
Single point with continuous latent representation (N dimensional vector)

Discrete geometry

Atomic structures are <u>hierarchical</u> and can be constructed from <u>recurring geometric motifs</u>.



We want to convert geometric information (3D coordinates of atomic positions)
into features on a trivial geometry (a single point)
and back again.



Discrete geometry

Single point with continuous latent representation (N dimensional vector)

Discrete geometry

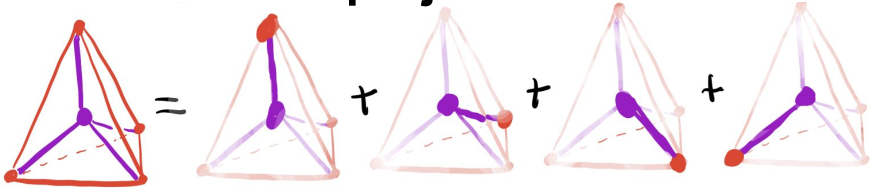
Atomic structures are <u>hierarchical</u> and can be constructed from <u>recurring geometric motifs</u>.

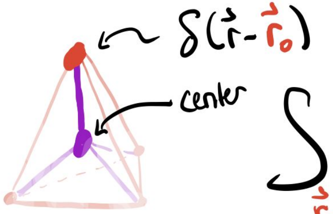
- Encode geometry
- Encode hierarchy

- Decode geometry
- Decode hierarchy

(Need to do this in a recursive manner)

To autoencode, we have to be able to convert *geometry* into *features* and *vice versa*.
We do this via spherical harmonic projections.





some choice of radial basis..

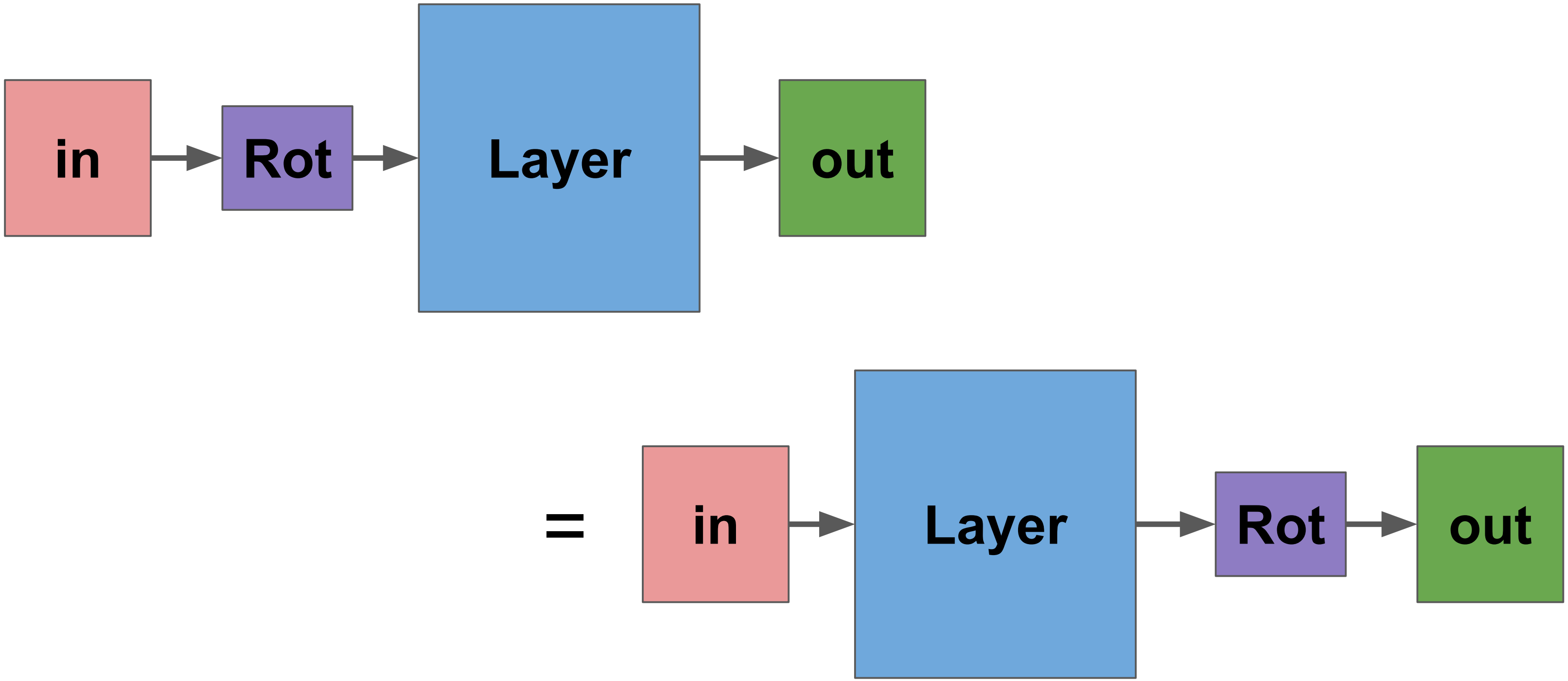
$$S_{\vec{r}_0} = \sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} \sum_{j=0}^{J} Y^l_m(\hat{r}_0)\, R_j(|\vec{r}_0|)$$

vector size $\left[\sum_{l=0}^{l_{max}} (2l+1) \times J\right]$

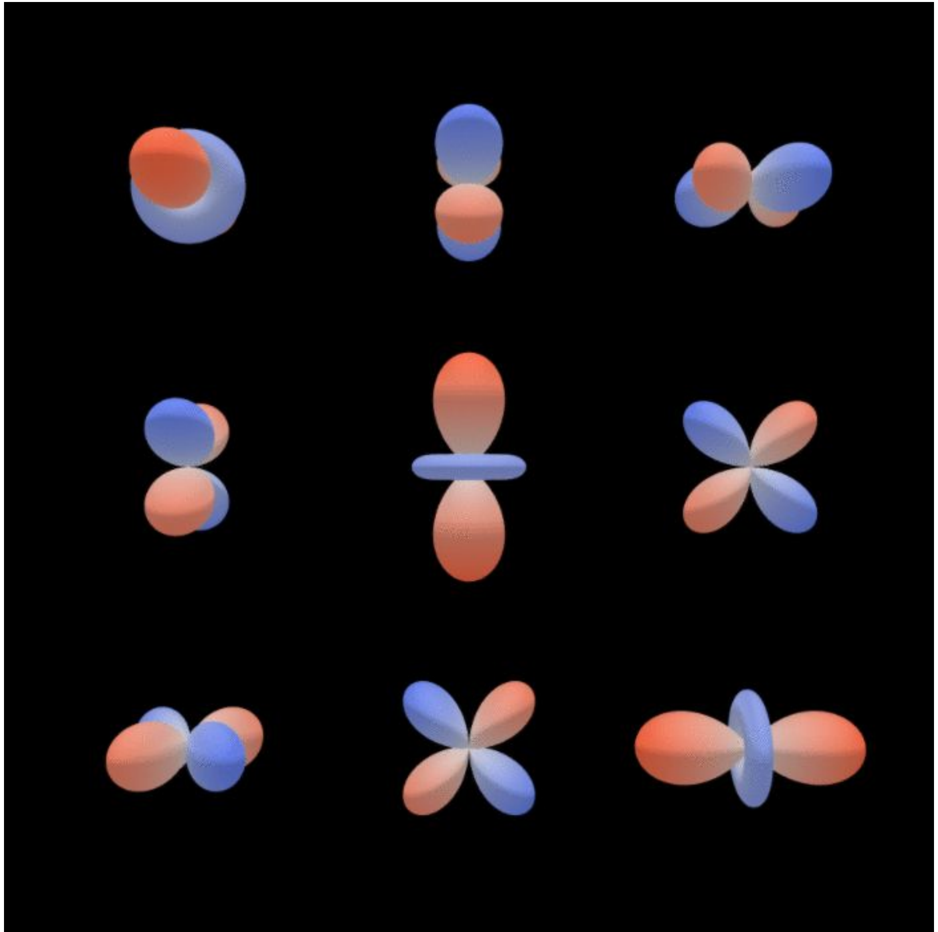
Yay! Linearity!

$$S_{tetrahedron} = S_{\vec{r}_0} + S_{\vec{r}_1} + S_{\vec{r}_2} + S_{\vec{r}_3}$$

To be rotation-equivariant means that we can rotate our inputs
<u>OR</u> rotate our outputs and we get the same answer ***(for every operation)*****.**



For L=1 ⇨ L=1, the filters will be a learned, radially-dependent linear combinations of the L = 0, 1, and 2 spherical harmonics.



Random filters for L=1 ⇨ L=1…

(3 in L=1 channels by 3 out L=1 channels)

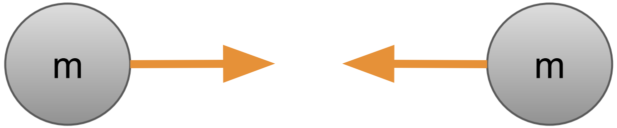
… as a function of increasing r.

Time showing filter for varying r, where $0 \leq r \leq r_{max}$.

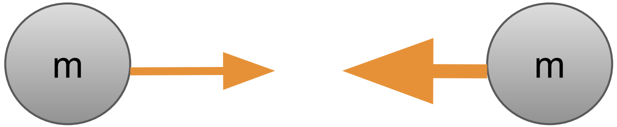
(+ / –) Radial distance is magnitude as a function of angle

Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

a.

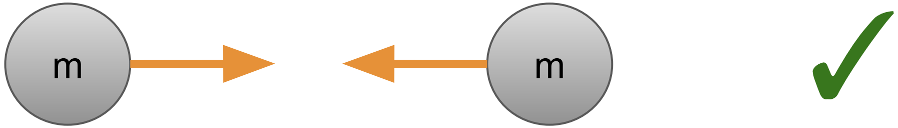


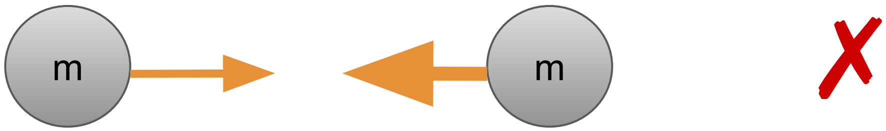
b.

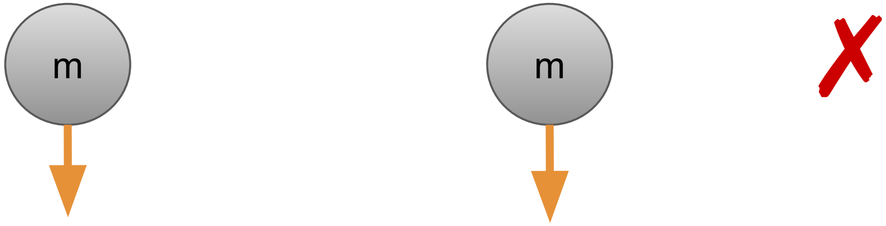


c.

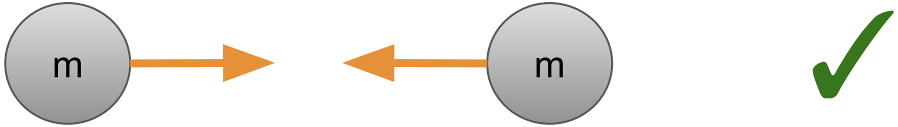
Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

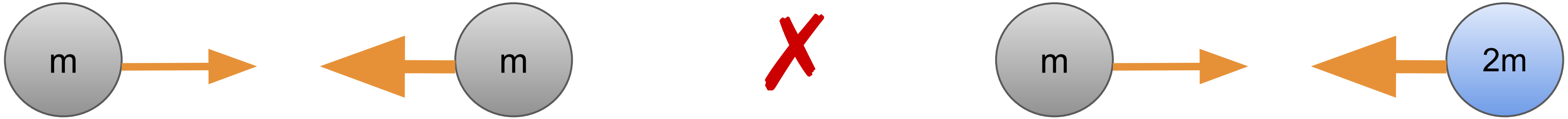
a. 

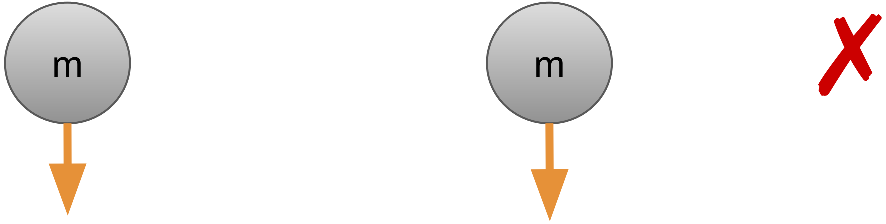
b. 

c. 

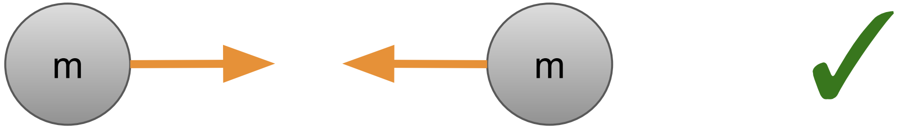
Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

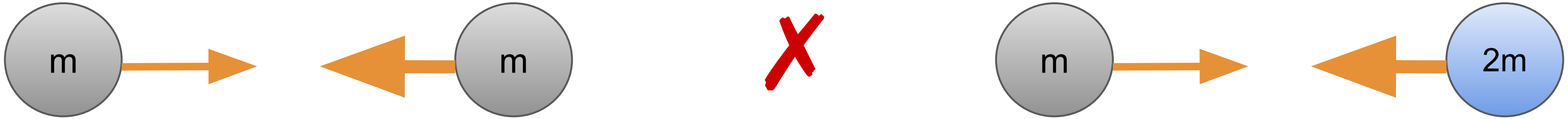
a.


b.


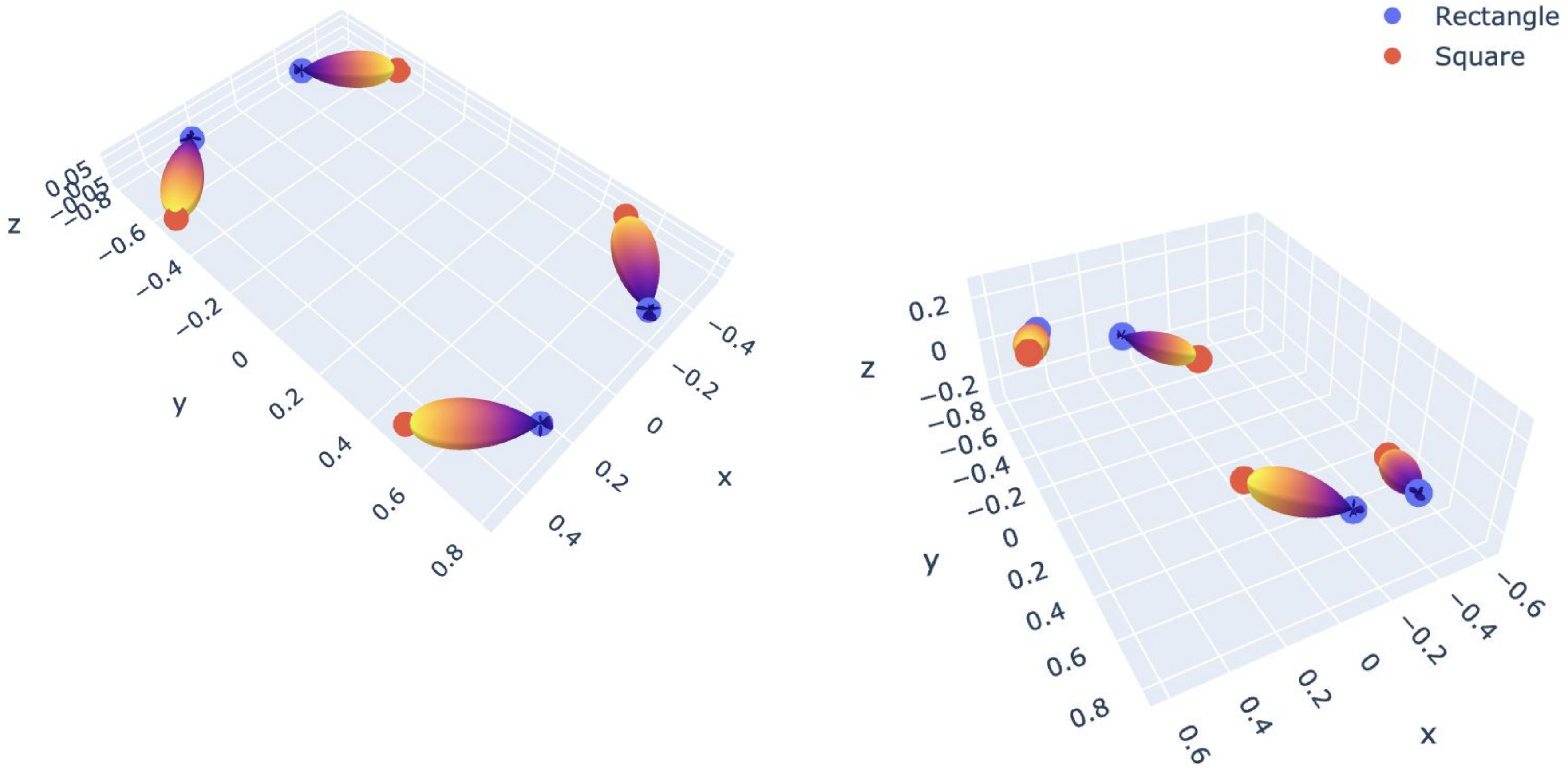
c.


Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

a.


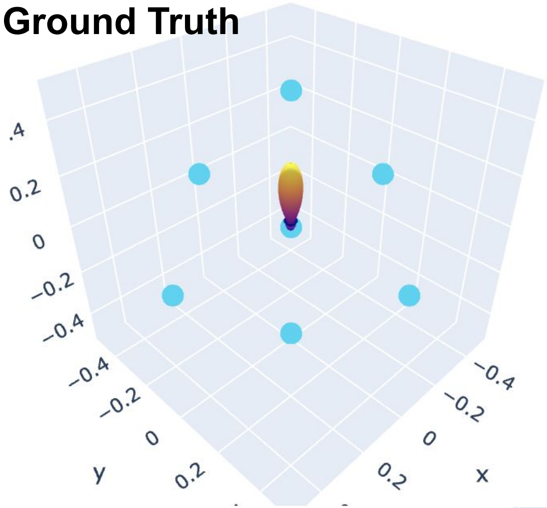
b.


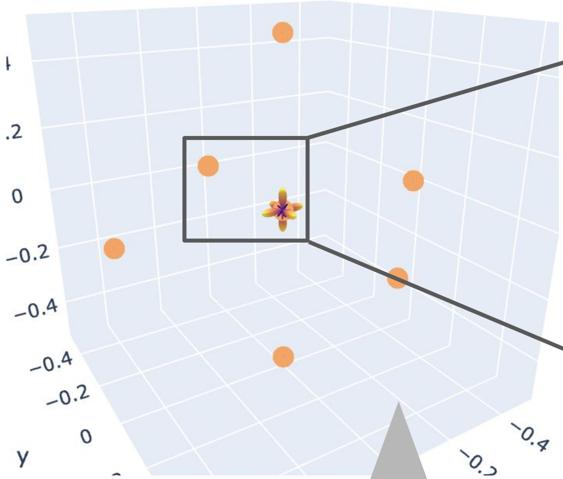
c.

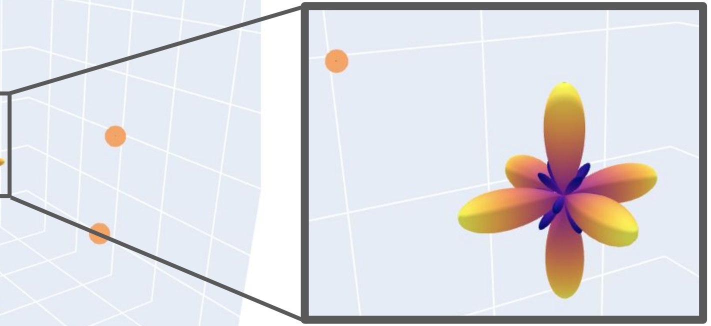


Rectangle
Square
z
y
x
0.05
−0.05
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
−0.4
−0.2
0
0.2
0.4
z
0.2
0
−0.2
y
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
x
−0.6
−0.4
−0.2
0
0.2
0.4
0.6

Predictions for O_h symmetry

Ground Truth

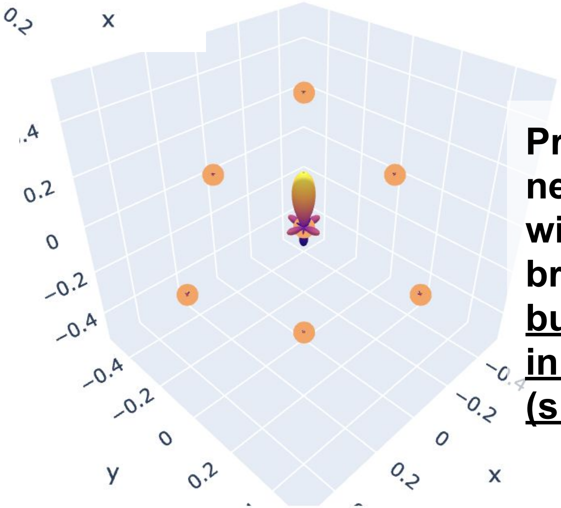






Superposition of 6 rotationally degenerate solutions.

Prediction of network trained with symmetry breaking input <u>and given symmetry breaking input along z.</u>



Prediction of network trained with symmetry breaking input <u>but given trivial input (single scalar)</u>.